

AFRL-VA-WP-TR-2002-3045

**AIR VEHICLE TECHNOLOGY
MODELING AND SIMULATION**

John Marino

**TechnoSoft, Inc.
4434 Carver Woods Drive
Cincinnati, OH 45242-5545**



DECEMBER 2001

FINAL REPORT FOR 25 SEPTEMBER 1998 – 25 SEPTEMBER 2000

THIS IS A SMALL BUSINESS TECHNOLOGY TRANSFER (STTR) PHASE 2 REPORT

Approved for public release; distribution is unlimited.

Copyright © 2000 by Alexander I. Danilin and Terrence A. Weisshaar

This material may be reproduced only by or for the U.S. Government pursuant to the copyright license in the contract.

**AIR VEHICLES DIRECTORATE
AIR FORCE MATERIEL COMMAND
AIR FORCE RESEARCH LABORATORY
WRIGHT-PATTERSON AIR FORCE BASE, OH 45433-7542**

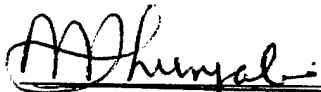
20020731 110

NOTICE

Using government drawings, specifications, or other data included in this document for any purpose other than government procurement does not in any way obligate the U.S. Government. The fact that the government formulated or supplied the drawings, specifications, or other data does not license the holder or any other person or corporation; or convey and rights or permission to manufacture, use, or sell any patented invention that may relate to them.

This report has been reviewed by the Office of Public Affairs (ASC/PA) and is releasable to the National Technical Information Service (NTIS). At NTIS, it will be available to the general public, including foreign nations.

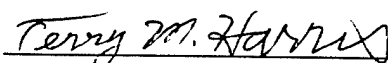
This technical report has been reviewed and is approved for publication.



Amarshi A. Bhungalia

Project Engineer

Structural Design & Development Branch



Terry Harris

Chief

Structural Design & Development Branch



DAVID M. PRATT, PhD

Technical Advisor

Structures Division

Copies of this report should not be returned unless return is required by security considerations, contractual obligations, or notice on a specific document.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

The public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Department of Defense, Washington Headquarters Services, Directorate for Information Operations and Reports (0704-0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to any penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number. PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.

| | | | | | | |
|---|------------------------------------|-------------------------------------|---|-----------------------------------|--|--|
| 1. REPORT DATE (DD-MM-YY) December 2001 | | | 2. REPORT TYPE Final | | 3. DATES COVERED (From - To) 09/25/1998 – 09/25/2000 | |
| 4. TITLE AND SUBTITLE Air Vehicle Technology Modeling and Simulation | | | | | 5a. CONTRACT NUMBER F33615-98-C-3208 | |
| | | | | | 5b. GRANT NUMBER | |
| | | | | | 5c. PROGRAM ELEMENT NUMBER 65502F | |
| 6. AUTHOR(S) John Marino | | | | | 5d. PROJECT NUMBER STTR | |
| | | | | | 5e. TASK NUMBER 03 | |
| | | | | | 5f. WORK UNIT NUMBER 01 | |
| 7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) TechnoSoft, Inc. 4434 Carver Woods Drive Cincinnati, OH 45242-5545 | | | | | 8. PERFORMING ORGANIZATION REPORT NUMBER | |
| 9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Air Vehicles Directorate Air Force Research Laboratory Air Force Materiel Command Wright-Patterson AFB, OH 45433-7542 | | | | | 10. SPONSORING/MONITORING AGENCY ACRONYM(S) AFRL/VASD | |
| | | | | | 11. SPONSORING/MONITORING AGENCY REPORT NUMBER(S) AFRL-VA-WP-TR-2002-3045 | |
| 12. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution is unlimited. | | | | | | |
| 13. SUPPLEMENTARY NOTES This is a Small Business Technology Transfer (STTR) Phase 2 report. This report contains copyright material. | | | | | | |
| 14. ABSTRACT <p>The aircraft design process is a complex, highly iterative, time-consuming process that contributes significantly to the overall engineering cost. The multidisciplinary nature of the engineering process, which includes design, analysis, and manufacturing, follows a regimented path initiated by a conceptual design, evolving into a preliminary design, leading to a detailed design for production. Many critical design decisions are made at the conceptual level where the least amount of information is available to assist in the design evaluation and tradeoffs.</p> <p>The objective of this effort is the development of an engineering framework to provide a natural environment for modeling and simulating the aircraft design process through its various states from the conceptual to preliminary phase. The framework will provide a modeling paradigm with an underlying object-oriented architecture enabling the seamless integration and automation of the various engineering processes.</p> <p>The framework supports a collaborative, highly interactive and creative design environment that enables disparate disciplinary efforts to concurrently share high fidelity object-oriented models. The system is implemented using Adoptive Modeling Language (AML) employing an advanced paradigm with dependency tracking and demand-driven computation, facilitating the feed forward and backward of information among the various engineering processes.</p> | | | | | | |
| 15. SUBJECT TERMS | | | | | | |
| 16. SECURITY CLASSIFICATION OF: | | | 17. LIMITATION OF ABSTRACT: SAR | 18. NUMBER OF PAGES 128 | 19a. NAME OF RESPONSIBLE PERSON (Monitor) Amarshi A. Bhungalia 19b. TELEPHONE NUMBER (Include Area Code) (937) 255-8335 | |
| a. REPORT Unclassified | b. ABSTRACT Unclassified | c. THIS PAGE Unclassified | | | | |

TABLE OF CONTENTS

| | |
|--|-----------|
| 1. FOREWORD | 1 |
| 2. SUMMARY | 2 |
| 3. DESIGN ARCHITECTURE AND FRAMEWORK..... | 2 |
| 4. TECHNICAL APPROACH..... | 3 |
| 4.1. AML FRAMEWORK | 4 |
| 4.2. GEOMETRY/COMPONENT CONFIGURATION BUILDER AND COMPLEX SURFACE MODELER | 6 |
| 4.3. FINITE ELEMENT INTERFACE TO PATRAN AND INNATE PANEL-BASED MESHER | 9 |
| 4.4. STRUCTURAL ANALYSIS INTERFACE WITH DRACON | 10 |
| 4.5. AERODYNAMIC ANALYSIS INTERFACES | 12 |
| <i>Missile Datcom</i> | 13 |
| <i>PANAIR</i> | 13 |
| 4.6. INTEGRATED ENVIRONMENT | 14 |
| APPENDICES..... | 15 |
| APPENDIX 1: MESH SURFACES AML DOCUMENTATION | 16 |
| APPENDIX 2: AML GENERAL AERODYNAMICS CLASSES | 23 |
| <i>Aerodynamic Coefficients Source Classes</i> | 23 |
| <i>Flight Conditions Class</i> | 36 |
| APPENDIX 3: AML/DATCOM INTERFACE DOCUMENTATION | 38 |
| APPENDIX 4: AML PANAIR INTERFACE DOCUMENTATION | 69 |
| <i>PANAIR Class with Common Aerodynamic Coefficients Interface</i> | 83 |
| APPENDIX 5: AML DRACON INTERFACE DOCUMENTATION | 86 |
| APPENDIX 6: GENERAL DESCRIPTION OF THE SANDWICH PLATE ELEMENT USED IN AML | 94 |
| <i>Stiffness matrix of skin panels</i> | 95 |
| <i>Skin panel stresses</i> | 102 |
| <i>Stiffness matrix of filler core</i> | 103 |
| APPENDIX 7: AML DRACON INTERFACE DOCUMENTATION THE USE OF OPTIMALITY CRITERIA FOR AIRCRAFT CONCEPTUAL LEVEL STRUCTURAL DESIGN | 106 |
| <i>Abstract</i> | 106 |
| <i>Background</i> | 106 |
| <i>Identification of aeroelastic characteristic (typical) sections</i> | 108 |
| <i>Stiffness constraints and optimality criteria</i> | 111 |
| <i>Example - vertical tail design with strength and Stiffness constraints</i> | 115 |
| <i>Conclusion</i> | 118 |
| <i>References</i> | 119 |

1. FOREWORD

This document provides information concerning development for the Air Force contract #F33615-98-C-3208 titled Air Vehicle Configuration. The Air Force Research Laboratory / Air Vehicle program monitor in this effort is Dr. Max Blair. TechnoSoft Inc. (TSI) provided the contracted effort as a prime contractor with Purdue University (PU) and Samara State University (SSU) serving as subcontractors. The Principle Investigators were as follows:

TSI: Adel Chemaly,

PU: Dr. Terrence Weisshaar,

SSU: Dr. Alexander Danilin and Dr. Valery Komarov.

The objective of this research was to develop a comprehensive design environment and framework for modeling and simulating aircraft systems that seamlessly integrates different engineering processes. This framework for Air Vehicle Configuration (AVC) covers not only the accurate definition of the external shape on which lift and drag depend, but it also includes the preliminary sizing of structural elements that tend to drive the weight of the vehicle and tend to influence the size of critical loads.

2. Summary

The repetitive, iterative nature of air vehicle design and the conflicting requirements for performance and affordability present a challenge to provide a comprehensive design system to integrate the various engineering stages. Capturing and modeling the engineering philosophies as related to the design evolution from concept to production details must be done to enable quick response to changes in designs, materials, and processes. Automating and integrating the various engineering cycles will speed the design process and provide high fidelity guidance for designers to interact across disciplinary barriers and to assess viability of the design and to define and to reduce economic risk. This document reports work performed regarding the aforementioned topics under Air Force STTR contract #F33615-98-C-3208 titled Air Vehicle Configuration.

The methodology of the developed AML modules is explained and documentation for the AML modules is given in the Appendices. The effort included integrating basic structural design capability into the AML aircraft design code being developed by Technosoft. Dr. Danilin worked with TechnoSoft personnel to integrate his DRACO code as part of the AML modules. The Appendices also include a summary of the finite element type used by Dr. Danilin in his AML work at TechnoSoft and a paper written by Dr. Weisshaar, based upon Dr. Danilin's work in design, in which the AML procedure is mentioned. The paper written by Dr. Weisshaar was presented at the 40th AIAA/Structural Dynamics and Materials Conference in Atlanta, Georgia in April 2000.

TechnoSoft, Purdue, and Dr. Danilin demonstrated examples of the technology to the Air Force Research Laboratory personnel in August 1999 with a sample fuselage bulkhead and wing design. Dr. Danilin and Dr. Komarov also submitted a description of their special sandwich element as required and also made themselves available for consultation.

Professor Weisshaar provided services while Dr. Danilin was in this country. He also helped in preparation of the final report and helped present results that showed the versatility of the methods used by AML.

3. Design Architecture and Framework

The objective of this research was the development of a comprehensive design environment and framework for modeling and simulating aircraft systems that seamlessly integrates different engineering processes. This framework covers not only the accurate definition of the external shape on which lift and drag depend and the arrangement of internal contents in a straightforward manner, but it also includes the preliminary sizing of structural elements that tend to drive the weight of the vehicle and tend to influence the size of critical loads.

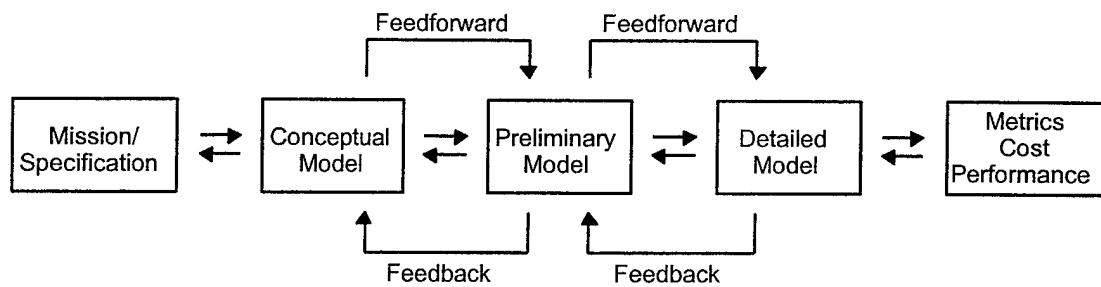


Figure 1: Design Process

Based on the Adaptive Modeling Language (AML), the system's framework supports a single underlying object-oriented architecture with demand-driven computation and dependency tracking that allows information to feed forward and backward among the various design levels and engineering processes as the design evolves as shown in Figure 1.

The system architecture is based on a modular framework; each module focuses on an engineering discipline or component design application. This supports the seamless integration of various tools and applications in the aircraft design process. An appealing feature of this architecture allows design tools to evolve or be updated without requiring changes to the interface within the architecture. Multiple tools may be made available to the designer that serve the same purpose. This will allow designers to select the appropriate level of analysis for their particular task. It will also allow for mixed design modes, with some analyses performed at a low-order conceptual level while others are done at a high fidelity, preliminary level.

The framework has two foci:

- 1) Constructing and linking together conceptual, preliminary, and detailed models,
- 2) Seamlessly integrating engineering processes and tools.

A limited prototype was developed to illustrate the design process; this prototype focuses on wing structure conceptual and preliminary design and analysis and its use was demonstrated. Preliminary versions of two modules were implemented. These modules focus on early geometric configurations and preliminary structural and aerodynamic analyses.

4. Technical Approach

The modeling of the air vehicle considered during the effort focused on the fuselage, inlets, and wing structure. This research/development effort produced a system framework, architecture, and a set of development modules to model and link strategies and engineering processes involved in aircraft conceptual and preliminary design. The framework provides an architecture and modules supporting an engineering environment

for the conceptual design, configuration, and first-order analysis of an air vehicle system. In addition, the system modules enable the modeling and automation of important engineering analysis processes to follow the design evolution from the conceptual to the preliminary stages. As a result, the system modules provide not only the basic capability for the conceptual design of an air vehicle, but also allow the design team to proceed to the design definition at a high level of fidelity, consistent with the requirements of major vehicle components and engineering analysis processes.

The framework involves a combination of low-level classes/methods/functions for air vehicle design, a common computational model for these classes, and an encompassing hierarchy for the classes to be used (instantiated) into an end-user environment. The following domain-specific modules have been developed:

1. Geometry/Component configuration builder and complex surface modeler
2. Finite element interface to Patran and innate panel-based mesher
3. Structural analysis interface with Dracon
4. Aerodynamic analysis interfaces

Following an elaboration on the underlying AML framework, each of these modules is described in the following sections along with a sample end-user environment demonstrating their use in an integrated framework for air vehicles. AML developer documentation is provided in the Appendices for several of these modules.

4.1. AML Framework

This work is based on the Adaptive Modeling Language (AML), offered commercially by TechnoSoft Inc., for knowledge-based concurrent engineering and comprehensive modeling that integrates design specifications, part geometry and features, manufacturing, inspection, and analysis processes in a unified part model. AML provides a KBE framework that captures knowledge from the modeled domain and creates parametric models with that knowledge. Classes inheriting from AML primitives may be defined and methods may be written against these classes providing user-defined behavior. After defining the classes, a hierarchical part model is initiated where the attributes of objects can be related using constraints. This part model may be used as a parametric design in a "what-if" scenario by changing design parameters and re-computing the model as the changes are propagated through the model on demand. In addition, AML is dimensionless and may be adapted to support any dimension units.

Various aspects of a problem can be detailed through a single unified model in AML. An example of such an application is structural design. A geometric design is created and various physical attributes are associated with the geometry. Then, the attributes for a finite element mesh and the knowledge required for generating input files for analysis is

maintained. AML allows this information to be stored in a single structured model. A complete user interface for the problem including input and output forms, menus, etc., can also be associated with the same part model that encompasses the various aspects of the application. The paradigm provides a common interface to a number of solid modelers in addition to different mesh generators and FEA solvers. The virtual layer interface provides a common, consistent interface to achieve this.

The proposed system is composed of several AML modules (sets of classes and methods) relating to the different knowledge domains and each performing a different function as required by the various engineering disciplines. All the modules in the system can be integrated within the AML object-oriented architecture through the Common Computational Model (CCM - see following paragraphs for further explanation), although they may communicate with external programs through the virtual layers. Since AML is modular, only the necessary systems need to be loaded into AML. Hence, if a problem requires a modeling framework but no graphics or geometry, only the kernel needs to be loaded for that application. Applications invoking different aspects of the proposed system built in AML use a common user interface to the system. Hence, user familiarization is required only with one interface irrespective of the applications.

The lowest level AML object provides the language constructs for defining classes, methods, and functions, as well as the creation of objects, constraints, and a tree hierarchy. All subsequent objects augment the language. A part/sub-part relation enables the creation of a tree-structured unified model where the children of any node of the tree represent sub-objects thus creating an organized data hierarchy. Various aspects of the problem can be structured hierarchically according to the domain being modeled (design, analysis, manufacturing, and inspection).

Each portion of this hierarchy supports AML's underlying constraint mechanism and demand-driven/dependency-backtracking behavior. Demand-driven means that the value of a property is not calculated until it is required (demanded). Until a value is demanded, an internal flag refers to the property's value as being unbound or the property as being smashed. Hence several properties that affect another property can be modified, but the affected property is not recalculated with each change; only when needed (demanded). Dependency tracking is the mechanism that propagates constraint changes throughout the part model. When a property is modified, all the properties it effects become unbound or smashed (not currently valid). When a property is smashed, it further smashes all properties that it effects, hence propagating the change by notifying entities that they need to be recalculated when next demanded.

Various applications of AML, including CAD, layout and configuration, CAM, and FEM/FEA, have different geometric requirements. These individual requirements are satisfied by augmenting the part model for different representations in order to satisfy the demands of various applications. These different representations are manipulated through a unified part model. AML presents a number of objects or classes for modeling complex geometrical operations in addition to simple primitives. Complex operations for mixed-dimensional solid/surface/wire frame Booleans, incorporating non-manifold topology, are

also supported. Additional objects for advanced modeling of free-form surfaces (NURBS, Beziers, etc.) are also available. The system supports IGES, STEP, and DXF interfaces to/from external CAD systems along with various graphic visualizations and geometric reasoning techniques.

AML provides a complete set of Graphical User Interface (GUI) classes, including forms, buttons, radio boxes, check boxes, input forms, and pop-up menus supported on Unix/Motif and Windows platforms. Since the user interface model is represented using the same knowledge representation system and syntax as the rest of the application, it too is dynamic in nature and the attributes of user interface entities can be altered dynamically depending on the model's state.

AML allows the integration for modules or programs that run outside of AML (external codes), through the use of wrapping objects and functions to allow for their access. The modules for conceptual design, geometry, and many of the first-order analyses were written directly in AML and integrated with external codes through AML's foreign function interface.

4.2. *Geometry/Component configuration builder and Complex surface modeler*

A major goal of the effort was the development and implementation of an engineering framework architecture that enables the layout, configuration and sizing of the major aircraft components. The focus was on a generic suite of classes and functionality to enable the designer to create and configure components in an organized fashion. This can then be applied specifically to a fuselage, wings, inlets, engines, and payload for example.

The implemented architecture serves as part of the AML object-oriented framework supporting a sophisticated interactive 3D graphical system that enables the manipulation of the sizing and position of vehicle components. The system also incorporates a 3D visualizer with support for high-end graphic functionality including interactive rotation, zooming, and rendering. The system enables the conceptual modeling of the vehicle using a layout of the vehicle's outer skin for a fuselage and wing using a series of constrained cross-sections and reference coordinate systems. These cross-sections' dimensions and orientations are dependent on certain conceptual vehicle parameters and additional user input. The system enables easy referencing and parametric association for feature properties that could be linked to external processes as a part of the virtual layer interface.

For layout and configuration, components may be placed anywhere within the model and their positions can be specified relative to other components or to the aircraft coordinate frame. Each component may have its own coordinate frame. Once the positions and orientations are specified and their dependencies set, moving the reference component will update the position of any dependent components. For example, some components, such as fuel tanks and ducts, can be specified by connecting cross-sections. These cross-sections are placed in the same manner as other components (using relative positions). In addition, the cross-sections of pre-existing components may be used. If any of the cross-

sections move (due to the changing position or size of the components to which they are anchored), then the duct-like components will automatically adjust.

Once the components are placed, their outer boundaries can be used to establish the constraints on the construction of the fuselage outer skin. Minimum clearance constraints can be imposed for sizing the outer surface model to allow for the subsystem layout and substructure sizing.

The wing geometry can be handled with the same type of cross-section configuration as the fuselage. In this case, the cross-sections can be airfoils. The wings are placed in the aircraft in the same way as fuselage components. The wings are divided into portions with each portion described by a root and tip airfoil and the wing can have properties such as twist, dihedral, and sweep.

The wing geometry can be handled with the same type of cross-section configuration as the fuselage. In this case, the cross-sections can be airfoils. The wings are placed in the aircraft in the same way as fuselage components. The wings are divided into portions with each portion described by a root and tip airfoil and the wing can have properties such as twist, dihedral, and sweep.

To better illustrate the process of building vehicle components an example of how a fuselage is constructed is described in the following paragraphs.

The user of the system specifies some basic fuselage properties (type of nose, length of nose, radius at nose, curvature at nose, ...). Using these properties a set of construction cross-sections and of scaling curves is generated. The cross-sections and the scaling curves are used to construct the fuselage's outer mold surface. The outer mold surface geometry contains mesh data that can be used to generate aero decks used by aero analysis codes. The strategy described above is used to build the wings and tails, and can be used to build any component geometry.

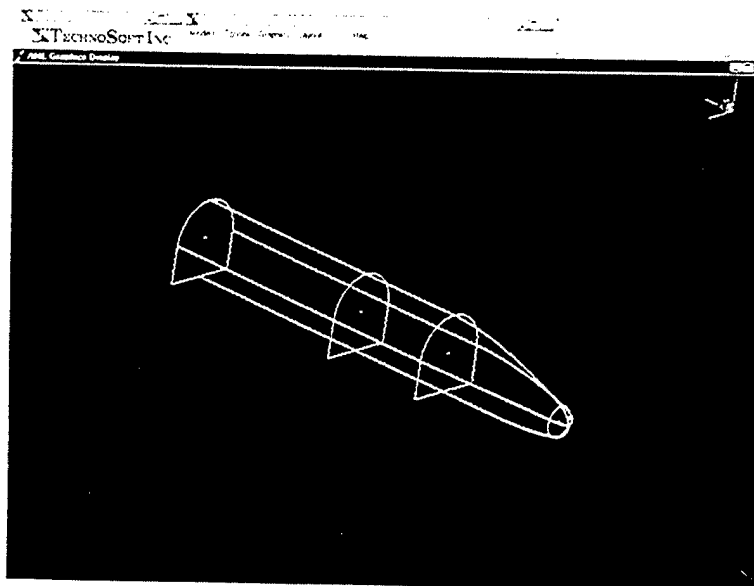


Figure 2: Fuselage Design

By defining the fuselage parameters the system automatically generates the cross-sections and the scaling curves as shown in Figure 2.

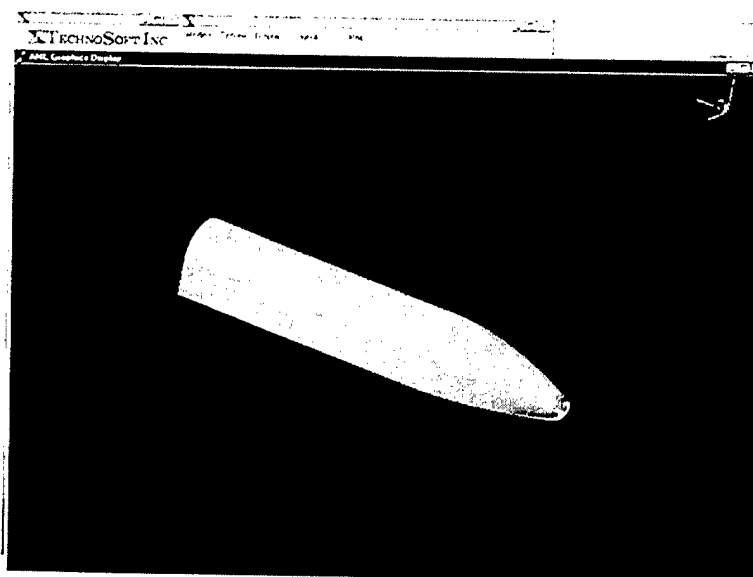


Figure 3: Fuselage Surface

Using the cross-sections and the scaling curves TechnoSoft Inc. has developed an effective and easy methodology for building a surface.

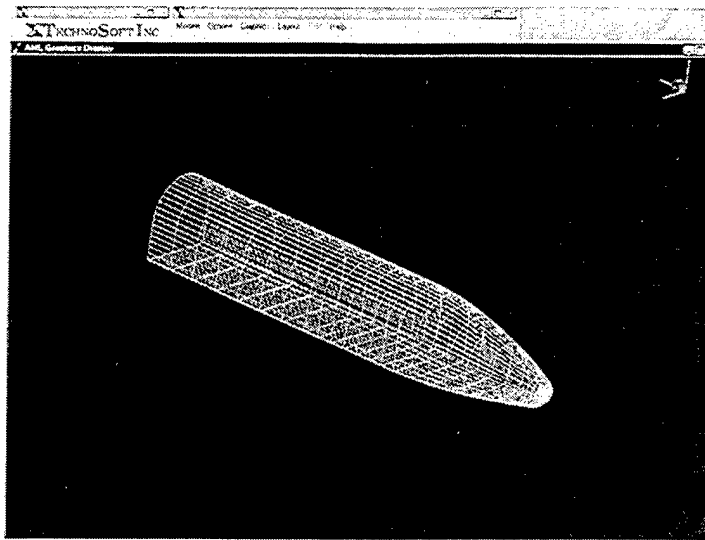


Figure 4: Mesh Surface

In addition to the outer mold surface, a quad mesh is generated as seen in Figure 4 allowing the user to generate data that can be used by the aero analysis codes.

For a list of AML functionality available in this module, please see Appendix 1.

4.3. *Finite element interface to Patran and innate panel-based mesher*

AML contains classes for attribute tagging and propagation to facilitate the association of information with entities in a geometric model. This information typically needs to be conveyed to downstream processes (manufacturing, inspection, meshing, or analysis). In a parametric modeling environment, reconfiguring a model involves modifying parameters at the construction level and regenerating the geometric model. Hence, all supplementary information would need to be conveyed to the final model as well as downstream processes every time the model is reconfigured. First, using attribute tagging, supplementary information is associated with configurable construction geometry. Next, every downstream operation, including final design, analysis, and manufacturing, has the information passed on through attribute propagation.

As a result, when the model is reconfigured (i.e., upstream design entities are modified in geometric or other properties), the attribute propagation mechanism ensures that supplementary information is passed downstream automatically. The attribute tagging and propagation mechanism is integrated with the demand-driven and backward-propagation mechanism using event properties.

The automatic mesh generation system is an AML module that allows tight integration of various mesh generation and analysis applications. The system provides a virtual interface to support various third-party mesh generators. The system permits selection of the geometry to mesh, tagging the vertices, edges, and faces of geometry for selective refinement of the mesh, and meshing the geometry by calling the external mesh generator.

It provides objects for meshing, as well as a user interface along with methods and a user interface for visualizing the mesh by querying the mesh database created by the mesh generator. This effort has extended AML's virtual layer interface to link with MSC's Patran mesher. Through the attribute tagging objects, attributes can be attached to the edges to initiate the seeding of the mesh. When changes are made to the model geometry, the finite element geometric model and all topology are automatically regenerated with all the appropriate meshing attributes, which is then passed on to the Patran mesh. Implementation focused on surface meshing since the finite element model requirement for the analysis of the air vehicle skins and substructure is completely made of shell elements.

Included in this new interface is the finite element analysis (FEA) system that enables the definition of an analysis problem by defining regions of interest, material models, solution strategies and other requirements for analyzing various problems using Patran and a finite element solver such as Nastran or Dracon. Various air vehicle components are modeled as AML classes that can be used to initiate a complete FEA problem model. The problem can be associated with the geometric objects as well as the mesh. The system generates several files that the solver can read and execute to generate results. Additionally, this system is modular and can be extended to provide a virtual solver layer that can talk to various other FEA solvers.

TechnoSoft has also created several innate paving algorithms within the framework architecture to automate the complete quad-panel mesh paving process. Classes have been developed to enable the user to model the strategy for the quad-panel models based on the parametric feature-based geometry and its associated material model. The quad-based finite element model is then automatically generated including all input for boundary conditions, loads distribution, etc., as required for a panel-based solver such as PANAIR.

For a list of AML functionality available in this module, please see Appendices 1 and 4.

4.4. *Structural analysis interface with Dracon*

The conceptual designer often relies on previous experience to ensure that sufficient space is provided for required structural members. Many times, the only consideration of structures during the conceptual design phase is related to weight estimation. This is not really structural design and this process subjects the conceptual design phase to undue risk. The statistical methods usually used in conceptual design stages limit design creativity and do not permit low risk investigation of radical new designs using new materials and manufacturing processes. In this system, the consideration of analysis includes a first-order analysis of critical loads and use of these loads to perform initial structural analysis for preliminary structural sizing and weight estimation.

Purdue and its partner Samara State worked with TechnoSoft to add functional capability to the AML code by developing a finite element analysis integration with the Dracon finite element solver and optimizer as well as a library of design elements that support

both low order modeling and cost/weight optimization at the conceptual level along with high fidelity modeling and cost/weight optimization at the preliminary design level. This modeling includes the ability to define the structural geometry/topology, its weight and stiffness, and to assess the quality of the aero/structural design. The models and methods used are proven techniques that have been developed scientifically overseas, but have not been captured in an effective way until now. The team examined computational issues and new scientific theories to develop numerical procedures to enable these techniques to be used with confidence in the AML environment.

The first-order analysis methods enable the initial layout of overall configuration and the sizing of substructures such as wings and bulkheads. Analysis links to higher-level analyses are also included. This finite element based analysis allow the modeling strategies to reduce the number of cycles required to decompose the model and generate the required next level finite element model. The framework architecture allows the designer to switch and combine levels of analysis (such as a fast, low-level aerodynamics solution combined with higher-level structural analysis).

The Dracon program has algorithms for structural layout and stiffening based on load paths and shape constraints. These algorithms have been tightly integrated with the overall framework to enable the layout and sizing of structure early in the conceptual design stage. These algorithms are based on solid theory following an innovative scientific process that still requires finite element models but does not require the computation power of a standard finite element solver and is focused toward the overall shape definition, sizing, and layout of the structures. This successful technology developed by Dr. Valery Komarov, Dr. Alexander Danilin, and their associates has been tested successfully at Purdue University by Professor Weisshaar and his group.

The Dracon analysis models can be integrated as part of the overall engineering framework. These models can be integrated with the concept model for providing the designer with feedback during the early design stages. Advanced visualization techniques have been developed to allow the presentation of the analysis results superimposed on the model geometry. The classes developed can be used for the output from Dracon as well as other solvers.

The functionality developed in AML Dracon module allow the user to choose from among several finite elements and lay out a preliminary structural design that can then be optimized to find a theoretically optimal structure and estimate the weight of this structure. This ability to optimize was not originally called for in the contract but was furnished by Dr. Danilin at no charge to the contract. This theoretically optimal structure will display Main Force Flows (MFF's' that are useful for the design stage).

In this development effort, the Samara team has developed/researched descriptions of theory and algorithms of structural and weight estimation for flat and wing-like structures with stress constraints, which include a set of finite elements with all necessary algorithms for the creation of stiffness matrices. These elements include:

- Classic rod
- Advanced membrane element
- Advanced shear element
- I-Beam element
- Special element sandwich element for FEM-I modeling (Danilin's own development)

A description of the sandwich element is given in the Appendix, while the other elements are standard elements available in the open literature and are thus not documented here.

An AML/DRACON module was developed and tested while Dr. Danilin was at TechnoSoft's facility. Dr. Danilin accomplished his work on an accelerated schedule and was finished in August 1999.

For a list of AML functionality available in this module, please see Appendix 5.

4.5. *Aerodynamic analysis interfaces*

TechnoSoft has developed, as well as extensively used, the aerodynamic analysis methodologies considered in this work. Work has been done to explore tools for the entire range of Mach number, dynamic pressure and vehicle configuration shape. A need exists to understand the use of these aerodynamic tools in developing data sets that have the appropriate level of detail for the calculations that are to be done. Table 1 lists the aerodynamics tools that were interfaced, with their respective capabilities.

| INDEPENDENT VARIABLE | AERODYNAMICS TOOL | |
|----------------------|------------------------------|----------------------------|
| | Missile DATCOM | PANAIR |
| Mach Number | 0.0 - 25.0 | 0 - 0.9999 1.0001 - 4.0 |
| Body Shape | Axis-symmetric Elliptical | Arbitrary |
| (Nose Type) | | |
| Subsonic: | Sharp | Arbitrary |
| Transonic: | Sharp | N/A |
| Supersonic: | Arbitrary | Arbitrary |
| Hypersonic: | Arbitrary | N/A |
| Angle of Attack | -180 to +180 deg. | Linear region |
| Sideslip Angle | -180 to +180 deg. | Linear region |
| Roll Angle | 0 to 360 deg. | 0 to 360 deg. |
| Altitude | Continuum Atmosphere | Continuum Atmosphere |

Table 1. Aerodynamics Analysis Tools Comparison

This suite of aerodynamic tools can be used for aerodynamic surface design and to create a total aero- thermodynamic profile for trajectory and structural-TPS analysis. Pre and post processing of aero thermal data for structural and thermal analysis and for displaying the results can be interfaced through PATRAN.

Missile Datcom

Missile DATCOM is an Air Force aerodynamic analysis code for a limited range of axis-symmetric and elliptical body missile configurations with multiple sets of lifting/control surfaces. This code uses slender body theory mixed with semi-empirical methodology to rapidly produce aerodynamic characteristics for subsonic, transonic, supersonic and hypersonic flight conditions. A key feature that has become very useful is the capability to input experimental component data (e.g., body, body-wing, body-tail) that is used in place of that component's code-calculated data.

Inputs are very simple geometry (especially as compared to panel codes), reference quantities and desired flight conditions. Outputs are 3-dimensional aerodynamic coefficients, including trim characteristics and static and dynamic derivatives.

Missile DATCOM was selected for integration into the system because of its ease of use and rapid and inexpensive calculations. The problem encountered for screening vehicles is that arbitrary bodies cannot be input using the code's geometry engine. This problem was solved by also integrating an aerodynamic panel code (PANAIR) to compute arbitrary body characteristics, processing its output into Missile DATCOM input format as "experimental" body alone component data, then using Missile DATCOM to generate complete configuration characteristics (a simple equivalent body of revolution is input for the Missile DATCOM body geometry). Use of the more CPU intensive panel code for the body alone calculations, serves to reduce turn-around time for the calculation of the aerodynamic characteristics and the preliminary screening of TAV configurations.

During the integration of Missile DATCOM, GUIs were developed for input and output processing. An algorithm to set up multiple runs of the code to develop and collect the aerodynamic output into coefficient tables of the appropriate format for direct input to trajectory codes was also created and implemented.

For a list of AML functionality available in this module, please see Appendices 2 and 3.

PANAIR

Higher-order panel methods offer the appropriate level of fidelity for force/moment analysis at acceptable run time for today's engineering workstation environment. The work developed allows the utilization of PANAIR (and QUADPAN in a later update) to capture trends of subsonic/transonic/supersonic pitching moments and drag-due-to-lift. Several other panel codes such as VORLAX, WOODWARD, APAS-UDP could be integrated in a similar methodology for developing panel models that are sufficiently robust for reliable trends as the vehicle shape is varied. In addition, the team has developed processes for vehicle trim analysis and drag-due-to-lift estimation that enhance the data fidelity of panel codes.

PANAIR is a NASA aerodynamic analysis code for arbitrary three-dimensional configurations in subsonic and supersonic speed regimes. The code solves the linearized

potential flow equations with a higher-order singularity distribution of sources and doublets over surface panels. In the past, the geometry input to this code has been very difficult, particularly at wing/body junctures, where it is easy for the user to leave a gap between panels. The PANAIR code contains procedures to "fix" these holes, but the whole process has been a large drawback. In cooperation with Lockheed Martin Orlando's Missiles and Fire Control Engineering Methods Group, PANAIR has been integrated into IMD, where it uses the geometry created within the IMD design environment to generate the surface elements, or paneling, at the push of a button. This paneling procedure leaves no gaps, and the panel normal vectors are always pointed outwards into the flow. The user can change the panel density and the panel-to-network relations, with the click of the mouse. Other PANAIR inputs, such as flight conditions, reference quantities, panel boundary conditions and wakes, have been assembled into user-friendly, hierarchical GUIs that automatically appear when related option buttons are clicked.

For a list of AML functionality available in this module, please see Appendix 2 and 4.

4.6. *Integrated Environment*

The system has been developed to allow the user to select which modules (and analyses) are used at any time. The user can also build models using the various modules and mix different levels of analysis. For example, a first-order aerodynamic analysis could be used to provide loads for a higher-order structural analysis.

TechnoSoft has used the Phase II resulting software modules into the underlying framework architecture for air vehicle design. This framework supports a collaborative design environment for seamlessly integrating various tools and engineering processes from the different disciplines. Lockheed Martin Missiles and Fire Control in Orlando and groups within the Air Vehicles Directorate of the Air Force Research Labs are currently using the developed modules and architecture in several AML-based applications.

Appendices

Appendix 1: Mesh Surfaces AML Documentation

This documentation is based on the premise that the cross sections used in these classes are in canonical space (created in the XY plane) and then oriented using a reference coordinate system.

| | |
|---|---------|
| mesh-surface-from-2-closed-cross-sections-class | [Class] |
|---|---------|

Allows the user to create a web surface provided curve-1-object and curve-2-object.

Inherit-from:

(mesh-points-from-2-closed-cross-sections-mixin mesh-surface-from-points-class)

Properties:

| | |
|--|--|
| curve-1-object | first curve |
| curve-2-object | second curve |
| curve-1-point-generation-method | Specify point generation method for curve-1. 'equal 'sin 3d-point-list<User provided coordinates>, or parameter-list |
| curve-2-point-generation-method | Specify point generation method for curve-2. 'equal 'sin 3d-point-list<User provided coordinates>, or parameter-list |
| split-curves-into-upper-lower? | flag to indicate whether to split input curves into upper and lower |
| number-of-points-per-cross-section-list | If input curves are to be split then the property is a list specifying the number of points on upper and lower curves, else it is a number specifying the number of points along the curves. |
| number-of-cross-sections | define the number of intermediate cross sections between curve-1-object and curve-2-object |
| intermediate-curves-shape-factor-list | Controls the shape of the intermediate sections (a factor of 0.0 maintains the shape of curve-1-object and factor of 1.0 maintains the shape of curve-2-object) |
| intermediate-curves-distance-factor-list | Controls the position of the intermediate curves between curve-1-object and curve-2-object |
| path-curve-object | Determines the path the intermediate curves follow to get from curve-1-object to curve-2-object |

| | |
|------------------------------|--|
| intermediate-alignment-type | Controls how the intermediate curves are aligned. 0 = curve-1-object normal, 1 = interpolate, 2 = follow path curve normal |
| x-plus-scaling-curve-object | points to the x-plus scaling curves |
| x-minus-scaling-curve-object | points to the x-minus scaling curves |
| y-plus-scaling-curve-object | points to the y-plus scaling curves |
| y-minus-scaling-curve-object | points to the y-minus scaling curves |

The user can query the object for the oriented mesh points by demanding the "combined-points" from an instance of MESH-SURFACE-FROM-2-CLOSED-CROSS-SECTIONS-CLASS. The points are oriented in such a way that the panel normals point outside.

| | |
|---|---------|
| mesh-surface-from-closed-cross-sections-class | [Class] |
|---|---------|

Inherit-from:

(mesh-points-from-closed-cross-sections-mixin mesh-surface-from-points-class)

Properties:

| | |
|--|---|
| cross-sections-list | cross sections to be meshed. |
| number-of-points-per-cross-section-list | Number of points per cross section |
| number-of-cross-sections-list | Number of intermediate cross sections |
| intermediate-curves-shape-factor-lists | A list of lists specifying the intermediate shapes factors |
| intermediate-curves-distance-factor-lists | A list of lists specifying the location of the intermediate curves. |
| path-curves-list | A list of path curves. Or the user can provide a single path curve. |
| intermediate-alignment-types-list | A list defining the alignment types. |
| cross-sections-point-generation-methods-list | Define point generation methods for curves. |
| split-cross-sections-into-upper-lower? | flag defining whether to split cross sections into upper and lower. |
| x-plus-scaling-curves-list | X-plus scaling curves. Can be a list or a single curve |
| x-minus-scaling-curves-list | X-minus scaling curves. Can be a list or a single curve |
| y-plus-scaling-curves-list | Y-plus scaling curves. Can be a list or a single curve |
| y-minus-scaling-curves-list | Y-minus scaling curves. Can be a list or a single curve |

The user can query the object for the oriented meshed points by demanding the "combined-points" from an instance of MESH-SURFACE-FROM-CLOSED-CROSS-SECTIONS-CLASS. The points are oriented in such a way that the panel normals point outside.

| | |
|---|---------|
| mesh-surface-from-2-open-cross-section-segments-class | [Class] |
|---|---------|

This class allows the user to create mesh (open) curve segments and build surfaces from the curve segments.

Inherit-from:

(mesh-surface-from-2-open-cross-section-segments-class)

Properties:

| | |
|--|--|
| curve-segment-list-1 | An ordered list of curve segments |
| curve-segment-list-2 | An ordered list of curve Segments |
| curve-segments-1-start-points-list | List of points listing the start points for seg1 in global space |
| curve-segments-2-start-points-list | List of points listing the start points for seg2 in global space |
| number-of-points-per-curve-segments-list | A list of numbers specifying the number of sampled points per segment. |
| number-of-cross-sections | Number of intermediate points. |
| intermediate-curves-shape-factor-list | A list of factors defining the shape. |
| intermediate-curves-distance-factor-list | A list of factors defining the position of the intermediate sections. |
| path-curve-object | The path the intermediate curves follow. |
| intermediate-alignment-type | Intermediate curves alignment types. 0 = curve-segment-list-1 normal, 1 = interpolate 2 = follow path curve normal |
| curve-segment-1-point-generation-method-list | point generation method for curves in curve-segment-list-1 |
| curve-segment-2-point-generation-method-list | point generation method for curves in curve-segment-list-2 |
| x-plus-scaling-curve-object | X-plus scaling curve object |

| | |
|------------------------------|------------------------------|
| x-minus-scaling-curve-object | X-minus scaling curve object |
| y-plus-scaling-curve-object | Y-plus scaling curve object |
| y-minus-scaling-curve-object | Y-minus scaling curve object |

For the user to get the oriented points associated with this object one needs to call GET-SHABP-OBJECTS passing an instance of MESH-SURFACE-FROM-2-OPEN-CROSS-SECTION-SEGMENTS-CLASS as input. This will return a list of instances. The user can then demand the combined-points from the returned instances.

| | |
|--|---------|
| mesh-points-from-open-cross-section-segments-mixin | [Class] |
|--|---------|

Inherit-from:

(basic-mesh-points-mixin)

Properties:

| | |
|--|--|
| curve-segment-lists | A list of lists defining the curve segments. |
| curve-segments-start-points-lists | A List of lists of points listing the start points for seg in global coord |
| number-of-points-per-cross-section-list | A list defining the number of points per segments. |
| number-of-cross-sections-list | A list defining the number of intermediate cross sections. |
| intermediate-curves-shape-factor-lists | A list of lists defining the shape factors. |
| intermediate-curves-distance-factor-lists | A list of lists defining the intermediate curves positions. |
| path-curves-list | A list defining the path curves. |
| intermediate-alignment-types-list | A list identifying the intermediate curves alignment type. |
| cross-sections-point-generation-methods-list | A list identifying point generation methods |
| x-plus-scaling-curves-list | X-plus scaling curve. Can be a list or a single curve. |
| x-minus-scaling-curves-list | X-minus scaling curve. Can be a list or a single curve. |
| y-plus-scaling-curves-list | Y-plus scaling curve. Can be a list or a single curve. |
| y-minus-scaling-curves-list | Y-minus scaling curve. Can be a list or a single curve. |

mesh-cap-from-closed-mesh-surface-class

[Class]

Inherit-from:

(mesh-surface-from-points-class)

Properties:

| | |
|----------------------------|--|
| mesh-surface-to-cap-object | The mesh surface to be capped. |
| front? | A flag set by the user indicating whether the cap lies at the front of the meshed surface or at the back of the surface. Used to determine the normal. |
| open? | Determines whether the cap is open or closed. |
| diameter | Diameter of cap, if cap is open. |
| flip-normal? | Allows the user to flip the cap normal. |
| center-point-coordinates | A global coordinate specifying where the cap is closed. |
| number-of-cross-sections | Number of intermediate curves. |

For the user to generate oriented-points the user needs to demand "COMBINED-POINTS" from an instance of type MESH-CAP-FROM-CLOSED-MESH-SURFACE-CLASS.

panel-normals-from-mesh-surface-points-class

[Class]

This class allows the user to draw the mesh-surface normals.

Inherit-from:

(simple-geometry-class)

Properties:

| | |
|----------------|---------------|
| mesh-surface | Mesh surface |
| normals-length | Normal length |

Allows the user to print the mesh data in a standard format

Inherit-from:

(object)

Properties:

| | |
|-----------------------------|---|
| mesh-surface | The surface to get the data from. |
| reference-coordinate-system | The user can define a reference coordinate system allowing the user to change the mesh point coordinates to reflect orientation of the reference coordinate system. |
| data-format | Mesh data can be formatted in a 'grid (cross sections), or 'panel format. |
| filename | The name of the mesh points data file |

Allows the user to read a mesh points data file, create the geometry, orient the geometry and then use an instance of MESH-SURFACE-DATA-CLASS to regenerate the mesh points data file.

Inherit-from:

(geom-object)

Properties:

| | |
|-----------------------------|---------------------------------------|
| filename | Data filename |
| reference-coordinate-system | Specify a reference coordinate system |

Allows the user to merge separate curve loops in a junction box.

Inherit-from:

(object)

Properties:

| | |
|-------------------------------|---|
| box-width | Junction box width |
| box-height | Junction box height |
| box-depth | Junction box depth |
| loops-to-join | A list of curve Loops to join. |
| start-node-number | The start node number |
| number-of-loop-points | A list of lists of box faces and number of points on the face edges. |
| number-of-intermediate-points | Number of intermediate cross sections between the loops to join and the box faces |

Appendix 2: AML General Aerodynamics Classes

This document covers the following class types:

- Aerodynamic Coefficient Sources
- Flight Conditions

The aerodynamic coefficients source classes provide a common interface for accessing coefficient data. Classes and applications can be written using interfaces that are independent of the source of coefficient information.

The flight conditions class provides a standard way to specify ranges of Mach numbers and flight angles.

Aerodynamic Coefficients Source Classes

These classes provide a common interface that can be used by all classes which provide aerodynamic coefficient data, whether that data originates from analysis, stored data in a file, or experimental data.

These classes assume that the data can be referenced in a multidimensional tabular form. The data is stored as lists of coefficient values and referenced by key values. The coefficients can be anything, but are typically quantities such as lift and drag coefficients. The keys are typically Mach numbers, angles of attack, etc. It is assumed that the data will be for ranges of key values (e.g., for Mach numbers 0.5, 0.6, 0.7, and 0.8) and that all combinations of key values will have corresponding coefficients.

The classes which provide this capability are *aerodynamic-coefficients-source-class* and

file-aerodynamic-coefficients-source-class. The class *aerodynamic-coefficients-plot-class* provides a way to present a graphical representation of the data contained in a class that inherits from *aerodynamic-coefficients-source-class*.

aerodynamic-coefficients-source-class

[Class]

This class is used to provide a common interface for other classes that compute aerodynamic coefficients. By combining this class with a class that computes the coefficients, any other class can access those coefficients using common properties and methods. This allows classes that require aerodynamic coefficients to be written without needing specific knowledge about the originating classes.

This class stores a multidimensional table of data. The data in the table is referenced by *keys*. These keys refer to things like Mach number or angle of attack. For each combination of key values, a set of coefficients can be retrieved from the table (if those key values are valid).

This class is typically used to provide access to coefficients that were generated by computing values for all possible combinations of key values (i.e., by looping through lists of Mach numbers, angles of attack, etc., and computing coefficients for each combination).

The keys and coefficients have associated symbols, labels, and descriptions. The symbol is a single-quoted name for the key or coefficient (e.g., 'M' for Mach number). The symbol is used for all reference to values of the key or coefficient. The label is a short string used to refer to the key or coefficient (e.g., "M" for Mach number). The label is used for short references. The description is a longer string used to refer to the key or coefficient (e.g., "Mach Number"). The description might be used in the labels for a graph or table.

Example user-specified property values:

| Property | Example Value |
|------------------------|---|
| ac-key-symbols | '(M alpha beta) |
| ac-key-ranges | '((0.5 0.7 0.9 1.1 1.3) (0.0 5.0 10.0) (0.0 2.0 4.0)) |
| ac-key-labels | '("M" "alpha" "beta") |
| ac-key-descriptions | '("Mach Number" "Angle of Attack" "Sideslip Angle") |
| ac-coefficient-symbols | '(CA CY CN CSL CSM CSN) |

The internally recognized coefficients are summarized in the following table. When the user specifies coefficients in *ac-coefficient-symbols*, the values in this table will be used to fill in the property *ac-coefficient-info*. The coefficients do not need to be restricted to those available in this table. If other coefficients are specified, either *ac-coefficient-info* will need to be overridden or the default values (using the symbol as label and description) will be used.

| Coefficient Symbol | Coefficient Label | Coefficient Description |
|--------------------|-------------------|----------------------------|
| CN | "CN" | "Normal Force Coefficient" |
| CSM | "Cm" | "Pitch Moment Coefficient" |
| CA | "CA" | "Axial Force Coefficient" |
| CY | "CY" | "Side Force Coefficient" |
| CSN | "Cn" | "Yaw Moment Coefficient" |
| CSL | "Cl" | "Roll Moment Coefficient" |
| CSMA | "Cma" | "dCm/dalpha" |
| CAN | "CNa" | "dCN/dalpha" |

| | | |
|------|---------|----------------------|
| CSNB | "Cnb" | "dCn/dbeta" |
| CYB | "CYb" | "dCY/dbeta" |
| CSLB | "Clb" | "dCl/dbeta" |
| CLCD | "CL/CD" | "CL/CD" |
| XCP | "Xcp" | "Center of Pressure" |
| CL | "CL" | "Lift Coefficient" |
| CD | "CD" | "Drag Coefficient" |

The coefficient data in this class should be accessed through the *get-coefficients* method.

Inherit-from:

(multi-key-table-class)

Properties:

| | |
|--------------------------|---|
| ac-key-symbols | A list of symbols used to refer to key values in table. |
| ac-coefficient-symbols | A list of symbols used to refer to coefficients in the table. |
| ac-key-ranges | A list of lists containing the values corresponding to each key in the table. |
| ac-key-labels | A list of short strings corresponding to each symbol in <i>ac-key-symbols</i> . |
| ac-key-descriptions | A list of strings providing descriptions of each symbol in <i>ac-key-symbols</i> . |
| ac-possible-coefficients | This property contains a list of lists of data providing default coefficient information. The contents of this list are summarized in the table above. This property should normally not be changed by the user. |
| ac-coefficient-info | This property contains a list of lists of information describing the coefficients listed in <i>ac-coefficient-symbols</i> . This property should normally not be changed by the user. This list will contain lists of (symbol label description) for each coefficient automatically created from the information in <i>ac-possible-coefficients</i> . The label and description will be string versions of the coefficient symbol for all symbols not available in <i>ac-possible-coefficients</i> . |

The following example class will be used to illustrate the methods that access the properties and data in this class. For the examples, assume that an instance of this class has been created and can be referenced as *!coeffs*.

```
(define-class example-aero-coeffs-class
  :inherit-from (aerodynamic-coefficients-source-class)
  :properties (
    ac-key-symbols      '(M alpha beta)
    ac-key-ranges       '((0.5 0.7 0.9) (0.0 5.0 10.0) (0.0 2.0))
    ac-key-labels       '("M" "alpha" "beta")
    ac-key-descriptions '("Mach Number" "Angle of Attack" "Sideslip Angle")
    ac-coefficient-symbols '(CA CY CN)

    ;The data in table-list would normally be generated by an analysis class.
    ;It is set directly in this example for illustration purposes only.
    table-list '(((0.5 0.0 0.0) (0.0 0.1 0.2))
                  ((0.5 0.0 2.0) (0.3 0.4 0.5))
                  ((0.5 5.0 0.0) (0.6 0.7 0.8))
                  ((0.5 5.0 2.0) (0.9 1.0 1.1))
                  ((0.5 10.0 0.0) (1.2 1.3 1.4))
                  ((0.5 10.0 2.0) (1.5 1.6 1.7))
                  ((0.7 0.0 0.0) (1.8 1.9 2.0))
                  ((0.7 0.0 2.0) (2.1 2.2 2.3))
                  ((0.7 5.0 0.0) (2.4 2.5 2.6))
                  ((0.7 5.0 2.0) (2.7 2.8 2.9))
                  ((0.7 10.0 0.0) (3.0 3.1 3.2))
                  ((0.7 10.0 2.0) (3.3 3.4 3.5))
                  ((0.9 0.0 0.0) (3.6 3.7 3.8))
                  ((0.9 0.0 2.0) (3.9 4.0 4.1))
                  ((0.9 5.0 0.0) (4.2 4.3 4.4))
                  ((0.9 5.0 2.0) (4.5 4.6 4.7))
                  ((0.9 10.0 0.0) (4.8 4.9 5.0))
                  ((0.9 10.0 2.0) (5.1 5.2 5.3))
                )
  )
)
```

get-key-range aerodynamic-coefficients-source-class

[Method]

This method returns the range of values for any of the keys.

Format:

(get-key-range instance symbol)

Arguments:

instance
symbol

An instance of the class *aerodynamic-coefficients-source-class*.
A symbol representing one of the keys to the data table.

Examples:

```
(get-key-range !coeffs 'M) returns '(0.5 0.7 0.9)
(get-key-range !coeffs 'B) returns nil since 'B is not a key for this data.
```

get-coefficients aerodynamic-coefficients-source-class

[Method]

This method should be used to access coefficient data in the *aerodynamic-coefficient-sources-class*.

This method returns a list of coefficients for specified key values. The *key-info* argument allows the user to request coefficient values for any combination of key values. The *coefficients* argument allows the user to request specific coefficients only.

The information is returned in the form of a list of lists. Each entry in the list contains a list of key values (in the order given in *key-info*) and a list of coefficient values (in the order given in *coefficients*).

Format:

```
(get-coefficients instance key-info coefficients)
```

Arguments:

| | |
|--------------|---|
| instance | An instance of the class <i>aerodynamic-coefficients-source-class</i> . |
| key-info | A list of lists containing key symbols and values. Coefficients which have keys with the specified values will be returned. In the results, the key values will be returned in the same order as specified in <i>key-info</i> . A range of <i>nil</i> indicates that all values in the table for that key should be returned. |
| coefficients | A list of requested coefficient symbols. The coefficients will be returned in the same order. |

Examples:

```

(get-coefficients !coeffs '((M (0.5)) (alpha (5.0)) (beta (0.0)))
  '(CA)) returns ((0.5 5.0 0.0) (0.6))

(get-coefficients !coeffs '((M) (alpha) (beta)) '(CA)) returns CA for
all values of 'M, 'alpha, and 'beta: '(((0.5 0.0 0.0)
(0.0)) ... ((0.9 10.0 2.0) (5.1))).

(get-coefficients !coeffs '((M (0.5)) (alpha (0.0 5.0)) (beta
(0.0))) '(CA)) returns (((0.5 0.0 0.0) (0.0))
((0.5 5.0 0.0) (0.6))).

(get-coefficients !coeffs '((M (0.5)) (alpha (0.0)) (beta (2.0)))
  '(CA CN CY)) returns (((0.5 0.0 2.0) (0.3 0.5
0.4)))

```

write-coefficients-file aerodynamic-coefficients-source-class

[Method]

This method is used to write the data contained in the *aerodynamic-coefficients-source-class* object to a file in a standard format. This format is common between all classes that inherit from *aerodynamic-coefficients-source-class*. Objects or applications written to read this format will be able to read data form any of these classes.

Format:

```
(write-coefficients-file instance filepath :coefficients)
```

Arguments:

| | |
|---------------|--|
| instance | An instance of the class <i>aerodynamic-coefficients-source-class</i> . |
| filepath | The full path and file name of the data file. |
| :coefficients | This argument allows the user to override the coefficient data that would normally be written. The default value is <i>nil</i> and this should not normally be changed. |

read-coefficients-file aerodynamic-coefficients-source-class

[Method]

This method is used to read key and coefficient data from a file written in the standard format (see the method *write-coefficients file*). This method returns a list of data – it does not directly change the object. This method is used by *file-aerodynamic-coefficients-file* to retrieve data.

Format:

```
(read-coefficients-file instance filepath)
```

Arguments:

| | |
|----------|---|
| instance | An instance of the class <i>aerodynamic-coefficients-source-class</i> . |
| filepath | The full path and file name of the data file. |

`generate-coefficients-file aerodynamic-coefficients-source-class` [Method]

This method is used to write the data contained in the *aerodynamic-coefficients-source-class* object to a stream in a standard file format. This file format is common between all classes that inherit from *aerodynamic-coefficients-source-class*. Objects or applications written to read this format will be able to read data from any of these classes. This method is called from within *write-coefficients-file*, which can be used to write the data directly to a file.

Format:

```
(generate-coefficients-file instance :stream :version  
                                :coefficients)
```

Arguments:

| | |
|---------------|--|
| instance | An instance of the class <i>aerodynamic-coefficients-source-class</i> . |
| :stream | A keyword specifying the stream that the data should be written to. The default value of <i>t</i> will write the data to the AML buffer. |
| :version | Specifies the file version that should be written. This will default to the latest version and should not normally be specified. |
| :coefficients | This argument allows the user to override the coefficient data that would normally be written. The default value is <i>nil</i> and this should not normally be changed. |

`generate-view-coefficients-table aerodynamic-coefficients-source-class` [Method]

This method is used to create a tabular report of the data contained in the *aerodynamic-coefficients-source-class*. This is primarily available to provide an easily human-readable report format.

Format:

```
(generate-view-coefficients-table instance :stream)
```

Arguments:

| | |
|----------|--|
| instance | An instance of the class aerodynamic-coefficients-source-class. |
| :stream | A keyword specifying the stream that the output should be written to. The default value of t will write the data to the AML buffer. |

| | |
|--|---------|
| file-aerodynamic-coefficients-source-class | [Class] |
|--|---------|

This class provides an interface to aerodynamic coefficient data that is contained in a file written in the standard format (i.e., by the method *write-coefficients-file*). Normally, this will be data that was created in another instance of a class inheriting from *aerodynamic-coefficients-source-class*.

To any other objects using the standard access methods (e.g., *get-coefficients*), this object will appear the same as the object where the data originated.

This class may also be used to provide an interface to externally created data (e.g., experimental data) which has been written to a file in the correct format.

Inherit-from:

```
(aerodynamic-coefficients-source-class)
```

Properties:

| | |
|-----------|---|
| filepath | The full path and file name of the data file. |
| file-info | This property contains information on the data in the file. It should not normally be accessed by the user and should not be changed. The normal <i>aerodynamic-coefficients-source-class</i> properties should be used to access the data. |

This class provides an x-y plot of the coefficient data contained in an instance of *aerodynamic-coefficient-source-class*.

The data can be plotted as lines of constant Mach number, constant angle of attack, or both. The values of the keys other than '*M*' and '*alpha*' must be constant for the plot and are set in the property *other-key-values*.

The appearance of the plot can be modified by changing the *datagraph-object* properties that are inherited into this class.

Inherit-from:

(datagraph-object)

Properties:

| | |
|---------------------|---|
| coefficients-object | This property should contain a reference to an instance of <i>aerodynamic-coefficient-source-class</i> . |
| ordinate-symbol | The symbol of the key or coefficient which should be used as the y-axis of the plot. This symbol must be present in either the <i>ac-key-symbols</i> or <i>ac-coefficient-symbols</i> property of the <i>coefficients-object</i> . |
| abscissa-symbol | The symbol of the key or coefficient which should be used as the x-axis of the plot. This symbol must be present in either the <i>ac-key-symbols</i> or <i>ac-coefficient-symbols</i> property of the <i>coefficients-object</i> . |
| show-mach-curves? | When <i>t</i> , lines of constant Mach number will be plotted. The default value is <i>t</i> . |
| show-alpha-curves? | When <i>t</i> , lines of constant angle of attack will be plotted. The default value is <i>t</i> . |
| other-key-values | A list of values for keys other than ' <i>M</i> ' and ' <i>alpha</i> '. These will be held constant in the plot. The order of the values corresponds to the order of the keys in <i>other-keys</i> . The default values will be the first entries in <i>ac-key-ranges</i> . |
| title-prefix | A string value in this property will be used as the beginning of the title displayed at the top of the graph. The default value is <i>nil</i> and indicates that no additional title text will be included. The title will |

be constructed from this property and the contents of *plot-description*.

| | |
|-----------------------------|--|
| key-symbols | A local copy of the property <i>ac-key-symbols</i> in <i>coefficients-object</i> . This should not be changed by the user. |
| coefficient-symbols | A local copy of the property <i>ac-coefficient-symbols</i> in <i>coefficients-object</i> . This should not be changed by the user. |
| mach-index | The position of the Mach number symbol in the list <i>ac-key-ranges</i> . This should not be changed by the user. |
| alpha-index | The position of the angle of attack symbol in the list <i>ac-key-ranges</i> . This should not be changed by the user. |
| mach-numbers | The range of Mach numbers in <i>ac-key-ranges</i> . This should not be changed by the user. |
| angles-of-attack | The range of angles of attack in <i>ac-key-ranges</i> . This should not be changed by the user. |
| other-keys | A list of keys other than 'M (Mach number) and 'alpha (angle of attack) in <i>coefficients-object</i> . This should not be changed by the user. |
| ac-key-descriptions | A local copy of the property <i>ac-key-descriptions</i> in <i>coefficients-object</i> . This should not be changed by the user. |
| ac-key-ranges | A local copy of the property <i>ac-key-ranges</i> in <i>coefficients-object</i> . This should not be changed by the user. |
| key-search-list | A list containing the key info required to get coefficients from <i>coefficients-object</i> . This should not be changed by the user. |
| coefficients-list | The results of a call to <i>get-coefficients</i> on <i>coefficients-object</i> . This should not be changed by the user. |
| variable-labels | A list of labels for all keys and coefficients in <i>coefficients-object</i> . This should not be changed by the user. |
| variable-descriptions | A list of descriptions for all keys and coefficients in <i>coefficients-object</i> . This should not be changed by the user. |
| variable-label-descriptions | A list of combined <i>variable-labels</i> and <i>variable-descriptions</i> . This should not be changed by the user. |
| mach-lists | A list of data used to draw constant Mach lines in the plot. This should not be changed by the user. |
| alpha-lists | A list of data used to draw constant angle of attack lines in the plot. This should not be changed by the user. |
| ordinate-index | The position of the <i>ordinate-symbol</i> in the <i>key-symbols</i> list. This should not be changed by the user. |
| abscissa-index | The position of the <i>abscissa-symbol</i> in the <i>key-symbols</i> list. This should not be changed by the user. |
| mach-curves | A list of data used to draw constant Mach lines in the plot. This should not be changed by the user. |
| alpha-curves | A list of data used to draw constant angle of attack lines in the plot. This should not be changed by the user. |
| data | The data that is drawn in the plot: a combination of <i>mach-curves</i> and <i>alpha-curves</i> as appropriate. This should not be changed by the user. |

multi-key-table-class

[Class]

This class provides basic functionality for maintaining a set of data in a table-lookup form. When using an instance of *aerodynamic-coefficients-source-class*, the access methods associated with that class should be used instead of those associated with *multi-key-table-class*.

Inherit-from:

(object)

Properties:

table-list

This property contains key-referenced data. It should **not** normally be changed or accessed by the user. The methods *add-record-to-table* and *get-records-from-table* should be used.

add-record-to-table multi-key-table-class

[Method]

This method is used to add data to an instance of *multi-key-table-class*.

Format:

(add-record-to-table instance keys value)

Arguments:

instance

An instance of the class *multi-key-table-class*.

keys

A list of the key values for the record being added to the object data.

value

A list of the value associated with the *keys* that is being added to the object data.

get-records-from-table multi-key-table-class

[Method]

This method is used to retrieve one or more records from an instance of *multi-key-table-class*. All records which match the specified keys will be returned.

Format:

(get-records-from-table instance keys)

Arguments:

instance
keys

An instance of the class *multi-key-table-class*.

A list of lists of key values for records which should be retrieved from the object. All records that match the values specified will be returned.

Flight Conditions Class

flight-conditions-class

[Class]

This class is used to specify a range of flight conditions that can be used as input to other objects (primarily analysis classes). Typically, by pointing to an instance of this class, these objects will automatically get Mach number and angle information. The analysis objects will then compute values (primarily aerodynamic coefficients) for each possible combination of Mach number and the angles. This behavior is from the analysis object – the *flight-conditions-class* is only used to specify input data.

This class has properties for two different modes for specifying flight angles. The mode is indicated by the property *angle-mode*. When *angle-mode* is *'body-axes'*, then the properties *angles-of-attack* and *sideslip-angles* should be used to specify a range of angles of attack and sideslip angles. When *angle-mode* is *'total'*, then the properties *total-angles-of-attack* and *roll-angles* should be used to specify a range of total angles of attack and roll-angles.

Some classes which use a reference to an instance of *flight-conditions-class* as an input may not be able to use both modes of flight angle specification.

Inherit-from:

(object)

Properties:

| | |
|------------------|---|
| mach-numbers | A list of Mach numbers. The default value is <i>nil</i> . |
| angle-mode | This property specifies which angle properties contain flight condition information. Allowed values are <i>'body-axes'</i> and <i>'total'</i> . When <i>angle-mode</i> is <i>'body-axes'</i> , the properties <i>angles-of-attack</i> and <i>sideslip-angles</i> should be used. When <i>angle-mode</i> is <i>'total'</i> , the properties <i>total-angles-of-attack</i> and <i>roll-angles</i> should be used. The default value is <i>'body-axes'</i> . |
| angles-of-attack | A list of angles of attack to be used when <i>angle-mode</i> is <i>'body-axes'</i> . The default value is <i>nil</i> . |

| | |
|------------------------|--|
| sideslip-angles | A list of sideslip angles to be used when <i>angle-mode</i> is 'body-axes'. The default value is <i>nil</i> . |
| total-angles-of-attack | A list of total angles of attack to be used when <i>angle-mode</i> is 'total'. The default value is <i>nil</i> . |
| roll-angles | A list of roll angles to be used when <i>angle-mode</i> is 'total'. The default value is <i>nil</i> . |

Appendix 3: AML/Datcom Interface Documentation

The *datcom-source-class* provides the primary interface to the Datcom application. Control of Datcom is through properties in this object and in its children. Additional objects may need to be instantiated to interface to the vehicle geometry.

All of the capabilities of this interface to Datcom can be accessed through properties. Users will normally not need to call any methods.

| |
|----------------------------|
| <i>datcom-source-class</i> |
|----------------------------|

| |
|---------|
| [Class] |
|---------|

The *datcom-source-class* provides an AML interface to Datcom.

Several of the properties of this class are used to point to instances of other classes. The *flight-conditions-object* property should point to an instance of class *flight-conditions-class*. If this property is given a value of nil, then the user must provide values for the flight condition properties (*flight-conditions-angle-mode*, *total-angles-of-attack*, *roll-angles*, *angles-of-attack*, *sideslip-angles*, and *mach-numbers*) as required (see the *flight-conditions-angle-mode* for property requirements).

Depending on the values given to some of the properties of this object, more than one run of Datcom may be required. This is handled transparently by the class; no user interaction or special setup is required.

The *datcom-source-class* has several subobjects, some of which have properties that should be changed by the user. In order to change the values or formulas of these properties, new classes will need to be created (inheriting from the originals) with replacement formulas and put in the subobjects section of a new datcom class inheriting from *datcom-source-class*. The subobjects, their classes, and whether they contain user-editable properties are summarized in this table:

| Subobject Name | Subobject Class | User Editable? |
|----------------|--|----------------|
| fltcon | datcom-fltcon-class | yes |
| refq | datcom-refq-class | yes |
| trim | datcom-trim-class | yes |
| finsets | datcom-finset-properties-class | no |
| body | datcom-body-properties-class | no |
| protuberances | datcom-protuberance-set-properties-class | no |
| inlet | datcom-inlet-properties-class | no |

This table summarizes the keys used to access the results of Datcom runs.

| Table Key | <i>ac-key-symbols</i> | <i>ac-key-labels</i> | <i>ac-key-descriptions</i> | Included when <i>flight-conditions-angle-mode</i> is |
|-----------------------|-----------------------|----------------------|----------------------------|--|
| Mach number | <i>'M</i> | "M" | "Mach Number" | |
| angle of attack | <i>'alpha</i> | "alpha" | "Angle of Attack" | <i>'body-axes</i> |
| sideslip angle | <i>'beta</i> | "beta" | "Sideslip Angle" | <i>'body-axes</i> |
| total angle of attack | <i>'alpha-t</i> | "alpha-t" | "Total Angle of Attack" | <i>'total</i> |
| roll angle | <i>'phi</i> | "phi" | "Roll Angle" | <i>'total</i> |

| | | | | |
|--------------------|---------------------------------------|---|--|--|
| deflection info | <i>'(delta wing-set wing)</i> | <i>'("delta" "wing-set" "wing")</i> | <i>'("Control Deflection" "Wing Set Number" "Wing Number")</i> | |
|--------------------|---------------------------------------|---|--|--|

This table summarizes the coefficients that are computed by Datcom and stored in this object. Coefficients are included only when their associated property is *t*.

| Coefficients | Included when |
|--|--|
| CN CSM CA CY CSN CSL CSMA CAN CSNB CYB CSLB CLCD XCP CL CD | always |
| CMQ CLNR CLLP | <i>compute-dynamic- derivatives?</i> |
| CAP | <i>include-power-on-data?</i> |
| CNT CAT CLT CDT | <i>include-trim-power-off- data?</i> |
| CAPT CDPT | <i>include-trim-power-on- data?</i> |

Inherit-From:

(aerodynamic-coefficients-source-class)

Properties:

flight-conditions-object The value of this property should be a *the*-reference to an instance of class *flight-conditions-class*. The datcom-source-class object will get values from the following properties from this object: *flight-*

conditions-angle-mode, total-angles-of-attack, roll-angles, angles-of-attack, sideslip-angles, and mach-numbers.

| | |
|--------------------------------|--|
| flight-conditions-angle-mode | The orientation of the vehicle can be specified using either the body axes angles or total angles. If the value is 'body-axes', then the properties <i>angles-of-attack</i> and <i>sideslip-angles</i> must be given values. If the value is 'total', then the properties <i>total-angles-of-attack</i> and <i>roll-angles</i> must be given values. If the <i>flight-conditions-object</i> property points to an instance of <i>flight-conditions-class</i> , then the value of <i>flight-conditions-angle-mode</i> will be taken from that object. Allowable values for <i>flight-conditions-angle-mode</i> are 'body-axes' and 'total'. |
| total-angles-of-attack | If the <i>flight-conditions-object</i> property does not point to a <i>flight-conditions-class</i> object and the <i>flight-conditions-angle-mode</i> property has a value of 'total', then this property should contain a list of the total angles of attack (in degrees) for which aerodynamic coefficients should be computed (e.g., '(0.0 5.0 10.0 15.0)'). |
| roll-angles | If the <i>flight-conditions-object</i> property does not point to a <i>flight-conditions-class</i> object and the <i>flight-conditions-angle-mode</i> property has a value of 'total', then this property should contain a list of the roll angles (in degrees) for which aerodynamic coefficients should be computed (e.g., '(0.0 2.0 4.0 6.0)'). |
| angles-of-attack | If the <i>flight-conditions-object</i> property does not point to a <i>flight-conditions-class</i> object and the <i>flight-conditions-angle-mode</i> property has a value of 'body-axes', then this property should contain a list of the angles of attack (in degrees) for which aerodynamic coefficients should be computed (e.g., '(0.0 5.0 10.0 15.0)'). |
| sideslip-angles | If the <i>flight-conditions-object</i> property does not point to a <i>flight-conditions-class</i> object and the <i>flight-conditions-angle-mode</i> property has a value of 'body-axes', then this property should contain a list of the sideslip angles (in degrees) for which aerodynamic coefficients should be computed (e.g., '(0.0 2.0 4.0 8.0)'). |
| mach-numbers | If the <i>flight-conditions-object</i> property does not point to a <i>flight-conditions-class</i> object, this property should contain a list of the Mach numbers for which aerodynamic coefficients should be computed (e.g., '(0.5 0.6 0.8 0.9 1.1 1.2)'). |
| fuselage-interface-object | This property should point to an instance of <i>datcom-fuselage-interface-class</i> . This object contains information required to create the Datcom input for the fuselage. |
| wing-set-geometry-objects | This property should contain a list of instances of <i>datcom-wing-set-interface-class</i> . Each of these instances will contain information required to create the Datcom input data for wings or fins. |
| protuberance-set-source-object | This property should point to a <i>datcom-protuberances-interface-class</i> . This object contains information required to create the Datcom input for protuberances. |
| inlet-source-object | This property should point to a <i>datcom-inlet-interface-class</i> . This object contains information required to create the Datcom input for the protuberance. |
| units-for-datcom-run | Determines the units of the data being given to and coming out of Datcom. Allowed values are 'in' 'ft' 'cm' 'm'. The default value is 'in'. |
| data-directory | The directory in which the output of the <i>datcom-source-class</i> object should be written. The default is (logical-path :temp). |

| | |
|------------------------------|---|
| results-file | The path and filename where the aerodynamic coefficients from the Datcom runs should be stored. The default file has the name "coeffs.dat" and will be written to the directory given by the property <i>data-directory</i> . |
| log-file | The path and filename where a record of the Datcom runs and output files should be stored. The default file has the name "datcom.log" and will be written to the directory given by the property <i>data-directory</i> . |
| output-directory | The directory in which the actual output files from Datcom should be written. Sequentially numbered copies of several of the output files will be kept in this directory (and logged in the file given by the property <i>log-file</i>). By default, this is the same directory as that given in the property <i>data-directory</i> . |
| compute-dynamic-derivatives? | If <i>t</i> , the values of the dynamic derivatives (CMQ, CLNR, CLLP) will be computed. If <i>nil</i> , no dynamic derivatives will be computed. |
| include-power-on-data? | If <i>t</i> , an additional aerodynamic coefficient will be computed for the power on case (CAP). If <i>nil</i> , this coefficient will not be computed. |
| include-trim-power-on-data? | If <i>t</i> , additional aerodynamic coefficients will be computed for the power on trim condition (CAPT, CDPT). If <i>nil</i> , no additional coefficients will be computed. |
| include-trim-power-off-data? | If <i>t</i> , additional aerodynamic coefficients will be computed for the power off trim condition (CNT, CAT, CLT, CDT). If <i>nil</i> , no additional coefficients will be computed. |
| run-fin-deflections? | If <i>t</i> , additional Datcom runs will be made to determine the aerodynamic coefficients for a range of fin deflections. The deflections are given by the wing or fin interface objects. |
| show-window-while-running? | If <i>t</i> , a window indicating that Datcom is running may be displayed. The default value is <i>nil</i> . |
| ac-key-symbols | A list of the keys used to access the results of the Datcom runs. The contents of this list are summarized in the table above and should not be changed by the user. |
| ac-key-labels | A list of printed names corresponding to the keys in <i>ac-key-symbols</i> . The contents of this list are summarized in the table above and should not be changed by the user. |
| ac-key-descriptions | A list of descriptive strings corresponding to the keys in <i>ac-key-symbols</i> . The contents of this list are summarized in the table above and should not be changed by the user. |
| deflection-key-info | A summary of key information reflecting control deflections. The contents of this list should not be changed by the user. |
| ac-key-ranges | A list of lists of the values which will be used for each key in Datcom. The entries in this list come from properties such as <i>mach-numbers</i> , <i>angles-of-attack</i> , etc. and should not be changed by the user. |
| ac-coefficient-symbols | A list of the symbols representing the data computed by Datcom and stored in <i>table-list</i> . The contents of this list is summarized in the table above and should not be changed by the user. |
| table-list | This property contains the cumulative results of all Datcom runs required to produce the requested data. This data is in the form of a list of lists. Each individual list contains a list of key values and a list of coefficient values. The order of the key and coefficient values corresponds to the order of the symbols in <i>ac-key-symbols</i> and <i>ac-coefficient-symbols</i> . |

methods

generate-datcom-file

run-datcom

datcom-fltcon-class

[Class]

This class is used as a subobject of *datcom-source-class*. When used as a subobject, the mach number and flight angle properties will, by default, get the value of the property in the parent *datcom-source-class*. They should not normally be overridden by the subobject.

Note: When *datcom-fltcon-class* is used as a subobject of *datcom-source-class*, the following properties will (by default), get their values from *datcom-source-class*: *mach-numbers*, *flight-conditions-angle-mode*, *angles-of-attack*, *sideslip-angles*, *total-angles-of-attack*, and *roll-angles*.

This class provides the entries for the FLTCON namelist in the Datcom input file.

| Datcom Entry | Corresponding Property | Datcom Entry | Corresponding Property |
|--------------|---|--------------|-------------------------|
| NALPHA | automatic | MACH | <i>mach-numbers</i> |
| ALPHA | <i>angles-of-attack, total-angles-of-attack</i> | REN | <i>reynolds-numbers</i> |
| BETA | <i>sideslip-angles</i> | ALT | <i>altitude</i> |
| PHI | <i>roll-angles</i> | HYPER | <i>newtonian-flow?</i> |

| | | | |
|-------|-----------|------|--------------------------------------|
| NMACH | automatic | SOSE | <i>second-order-shock-expansion?</i> |
|-------|-----------|------|--------------------------------------|

Inherit-From:

(object)

Properties:

| | |
|------------------------------|--|
| mach-numbers | A list of the Mach numbers for which aerodynamic coefficients should be computed (e.g., '(0.7 0.8 0.9 1.2 1.5)'). The formula of this property should normally not be overridden. The default formula is (default nil). |
| flight-conditions-angle-mode | The orientation of the vehicle can be specified using either the body axes angles or total angles. If the value is ' <i>body-axes</i> ', then the properties <i>angles-of-attack</i> and <i>sideslip-angles</i> must be given values. If the value is ' <i>total</i> ', then the properties <i>total-angles-of-attack</i> and <i>roll-angles</i> must be given values. If the <i>flight-conditions-object</i> property points to an instance of <i>flight-conditions-class</i> , then the value of <i>flight-conditions-angle-mode</i> will be taken from that object. Allowable values for <i>flight-conditions-angle-mode</i> are ' <i>body-axes</i> ' and ' <i>total</i> '. The formula of this property should normally not be overridden. The default formula is (default ' <i>body-axes</i> '). |
| angles-of-attack | A list of the angles of attack for which aerodynamic coefficients should be computed (e.g., '(0.0 5.0 10.0 15.0)'). This property will only be used when <i>flight-conditions-angle-mode</i> is ' <i>body-axes</i> '. The formula of this property should normally not be overridden. The default formula is (default nil). |
| sideslip-angles | A list of the sideslip angles (in degrees) for which aerodynamic coefficients should be computed (e.g., '(0.0 2.0 4.0 8.0)'). This property will only be used when <i>flight-conditions-angle-mode</i> is ' <i>body-axes</i> '. The formula of this property should normally not be overridden. The default formula is (default nil). |
| total-angles-of-attack | A list of the total angles of attack (in degrees) for which aerodynamic coefficients should be computed (e.g., '(0.0 5.0 10.0 15.0)'). This property will only be used when <i>flight-conditions-angle-mode</i> is ' <i>total</i> '. The formula of this property should normally not be overridden. The default formula is (default nil). |
| roll-angles | A list of the roll angles (in degrees) for which aerodynamic coefficients should be computed (e.g., '(0.0 2.0 4.0 6.0)'). This property will only be used when <i>flight-conditions-angle-mode</i> is ' <i>total</i> '. The formula of this property should normally not be overridden. The default formula is (default nil). |
| conditions-specifier | Flight conditions can be specified for a range of altitudes or Reynold's numbers. When <i>conditions-specifier</i> has a value of ' <i>ren</i> ', the <i>reynolds-numbers</i> property should contain a list of Reynold's numbers for which aerodynamic coefficients will be computed. |

| | |
|-------------------------------|--|
| reynolds-numbers | When <i>conditions-specifier</i> has a value of 'alt', the <i>altitude</i> property should contain a list of altitudes for which aerodynamic coefficients will be computed. The default value is 'alt'. When <i>conditions-specifier</i> has a value of 'ren', then this property should contain a list of Reynold's numbers at which aerodynamic coefficients should be computed. The number of Reynold's numbers specified should correspond to the number of <i>mach-numbers</i> . The default is a list with all elements equal to 1E6. |
| altitude | When <i>conditions-specifier</i> has a value of 'alt', then this property should contain a list of altitudes for which aerodynamic coefficients will be computed. The number of altitudes specified should correspond to the number of <i>mach-numbers</i> . The default is a list with all elements equal to 0.0. |
| second-order-shock-expansion? | When the value of this property is <i>t</i> , the SOSE card will be included in the Datcom input file. When <i>nil</i> , it will not. |
| newtonian-flow? | When the value of this property is <i>t</i> , the HYPER card will be included in the Datcom input file. When <i>nil</i> , it will not. |

datcom-refq-class

[Class]

This class is used as a subobject of *datcom-source-class*.

This class provides the entries for the REFQ namelist in the Datcom input file.

| Datcom Entry | Corresponding Property | Datcom Entry | Corresponding Property |
|--------------|--------------------------------------|--------------|--|
| SREF | <i>reference-area</i> | XCG | <i>cg-position-longitudinal, trim-cg-position-longitudinal</i> |
| LREF | <i>reference-length-longitudinal</i> | ZCG | <i>cg-position-vertical</i> |
| LATREF | <i>reference-length-lateral</i> | SCALE | <i>scale-factor</i> |
| ROUGH | <i>surface-roughness</i> | BLAYER | <i>boundary-layer-type</i> |
| RHR | <i>roughness-rating</i> | | |

Inherit-From:

(object)

Properties:

| | |
|-------------------------------|--|
| fuselage-object | This property should point to an instance of class <i>datcom-fuselage-interface-class</i> . Normally, this is done by getting its value from the <i>fuselage-interface-object</i> property of its parent, the <i>datcom-fuselage-interface-class</i> . The formula for this property should be \sim <i>fuselage-interface-object</i> . |
| reference-area | The reference area of the vehicle. When <i>fuselage-object</i> points to an instance of <i>datcom-fuselage-interface-class</i> , this value will be found automatically from that object. The formula of this property should normally not be overridden. |
| reference-length-longitudinal | The vehicle longitudinal reference length. When <i>fuselage-object</i> points to an instance of <i>datcom-fuselage-interface-class</i> , this value will be found automatically from that object. The formula of this property should normally not be overridden. |
| reference-length-lateral | The vehicle lateral reference length. When <i>fuselage-object</i> points to an instance of <i>datcom-fuselage-interface-class</i> , this value will be found automatically from that object. The formula of this property should normally not be overridden. |
| roughness-specifier | Specifies whether the <i>surface-roughness</i> or <i>roughness-rating</i> property will be sent to Datcom. When <i>roughness-specifier</i> is 'rough', the <i>surface-roughness</i> will be given. When <i>roughness-specifier</i> is 'rhr', the <i>roughness-rating</i> will be given. The default value is 'rough'. |
| surface-roughness | A value for the vehicle surface roughness. This will only be written when <i>roughness-specifier</i> is 'rough'. The default value is 0.0. |
| roughness-rating | A value for the vehicle roughness rating. This will only be written when <i>roughness-specifier</i> is 'rhr'. The default value is 0.0. |
| cg-position-longitudinal | The longitudinal position of the vehicle center of gravity. This value will be written when coefficients are computed in non-trim cases. The default value is 0.0. |
| trim-cg-position-longitudinal | The longitudinal position of the vehicle center of gravity under trim conditions. This value will be written when trim coefficients are computed. The default value is 52% of the vehicle length when <i>fuselage-object</i> is specified and 0.0 when it is not. |
| cg-position-vertical | The vertical position of the vehicle center of gravity. The same value will be used for trim and non-trim conditions. The default value is 0.0. |
| scale-factor | A scale factor used to multiply vehicle dimensions (other than reference dimensions). The default value is 1.0. |
| boundary-layer-type | A specifier for the type of boundary conditions to be used. The available options are 'turb' and 'natural'. The default value is 'turb'. |

This class is used as a subobject of *datcom-source-class*.

This class provides the entries for the TRIM namelist in the Datcom input file.

| Datcom Entry | Corresponding Property | Datcom Entry | Corresponding Property |
|--------------|-----------------------------------|--------------|---------------------------|
| SET | <i>trim-fin-set-object</i> | DELMIN | <i>minimum-deflection</i> |
| PANL | <i>use-panel-flags-list</i> | DELMAX | <i>maximum-deflection</i> |
| ASYM | <i>flip-panel-deflection-list</i> | | |

Inherit-From:

(object)

Properties:

| | |
|---------------------------------|---|
| datcom-finset-properties-object | This property should point to an instance of <i>datcom-finset-properties-class</i> . Normally, this should point to the <i>finsets</i> subobject of the <i>datcom-source-class</i> object. The formula for this property should be $\wedge finsets$. |
| minimum-deflection | The minimum deflection value allowed for fins during trim computations. The default value is -25.0. |
| maximum-deflection | The maximum deflection value allowed for fins during trim computations. The default value is 25.0. |
| use-panel-flags-list | A list of values (either <i>t</i> or <i>nil</i>) which indicate the panels of a finset which that can be deflected during trim computations. The default value will be a list with all <i>t</i> elements (all panels can be deflected). |
| flip-panel-deflection-list | A list of values (either <i>t</i> or <i>nil</i>) which indicate whether the sign of deflections should be reversed. The default value will be a list of all <i>nil</i> elements (all panels with normal deflection). |

This class is used as a subobject of *datcom-source-class*.

This class acts as an intermediate object between the wing descriptions in instances of *datcom-wing-set-interface-class* and Datcom. Normally, nothing should be changed in this class.

This class provides the entries for the FINSET namelists in the Datcom input file. The actual values are obtained from the *datcom-wing-set-interface-class* objects referenced by *source-wing-sets*.

Inherit-From:

(object)

Properties:

- | | |
|-------------------------|--|
| source-wing-sets | This property should contain a list of references to instances of <i>datcom-wing-set-interface-class</i> . Normally, this is done by getting its value from the <i>wing-set-geometry-objects</i> property of its parent, the <i>datcom-source-class</i> . The formula of this property should be \wedge <i>wing-set-geometry-objects</i> . |
| sorted-source-wing-sets | A list of the instances specified in <i>source-wing-sets</i> in order from vehicle nose to tail. This should not be changed by the user. |

This class is used as a subobject of *datcom-source-class*.

This class acts as an intermediate object between the fuselage description in an instance of *datcom-fuselage-interface-class* and Datcom. Normally, nothing should be changed in

this class.

The information written to the Datcom input file will either be geometry based or from aerodynamic coefficients depending on the settings of the properties. Control of this behavior should be through the properties of the *datcom-fuselage-interface-class* instead of this object.

This class provides the entries for the ELLBOD or AXIBOD namelist in the Datcom input file.

Inherit-From:

(object)

Properties:

- fuselage-interface-object* This property should point to an instance of *datcom-fuselage-interface-class*. Normally, this is done by getting its value from the *fuselage-interface-object* property of its parent, the *datcom-source-class*. The formula of this property should be \sim *fuselage-interface-object*.
- fuselage-aero-coeffs-source-object* This property should point to an instance of *aerodynamic-coefficients-source-class* if aerodynamic coefficients describing the body should be used instead of a geometric description. By default, the value of this property comes from the object pointed to by *fuselage-interface-object* and it should not be changed by the user.

datcom-protuberance-set-properties-class

[Class]

This class is used as a subobject of *datcom-source-class*.

This class acts as an intermediate object between the protuberance set descriptions in an instance of *datcom-protuberances-interface-class* and Datcom. Normally, nothing should

be changed in this class.

This class provides the entries for the PROTUB namelist in the Datcom input file.

Inherit-From:

(object)

Properties:

protuberance-set-source-object This property should point to an instance of *datcom-protuberances-interface-class*. Normally, this is done by getting its value from the *protuberance-set-source-object* property of its parent, the *datcom-source-class*. The formula of this property should be $\wedge protuberance-set-source-object$.

datcom-inlet-properties-class

[Class]

This class is used as a subobject of *datcom-source-class*.

This class acts as an intermediate object between the inlet description in an instance of *datcom-inlet-interface-class* and Datcom. Normally, nothing should be changed in this class.

This class provides the entries for the INLET namelist in the Datcom input file.

Inherit-From:

(object)

Properties:

| | |
|---------------------|--|
| inlet-source-object | This property should point to an instance of <i>datcom-inlet-interface-class</i> . Normally, this is done by getting its value from the <i>inlet-source-object</i> property of its parent, the <i>datcom-source-class</i> . The formula of this property should be \wedge <i>inlet-source-object</i> . |
|---------------------|--|

datcom-fuselage-interface-class

[Class]

This class is used to provide an interface between fuselage information and Datcom. All input required for Datcom can be specified with properties of this object. Additional methods for providing information are also provided.

The information provided to Datcom to describe the vehicle fuselage can be either geometric or aerodynamic. In the case of geometric data, properties describing the shape of the nose, center, and aft sections of the fuselage must be provided. For the case of aerodynamic data, a reference to an instance of *aerodynamic-coefficients-source-class* must be provided. If coefficients are to be provided, the property *aero-coeffs-source-object* must point to an instance of *aerodynamic-coefficients-source-class*. The geometric properties will be ignored. If *aero-coeffs-source-object* is nil, then the geometric properties will be used.

When the fuselage description is based on geometry, the actual properties describing the nose, center, and aft section geometry are contained in subobjects of *datcom-fuselage-interface-class*: *nose-info*, *center-body-info*, and *aft-body-info*. The *datcom-input* properties of *datcom-fuselage-interface-class* get their values from the corresponding properties of these subobjects.

Reference Properties:

The reference properties *reference-area*, *reference-diameter-longitude*, *reference-diameter-latitude*, and *body-length* may have their values computed directly from geometry; found from a *body-geometry-interface-class* object; found based on the

properties of the nose, body, and aft subobjects; or entered directly as values. If the *geometry-source-object* property points to an object of class *body-geometry-interface-class*, then the reference properties will be taken from that object. When the *geometry-source-object* property points to an object with a geom, the reference properties will be determined from that geom if the *cs-info* property is not *nil* (see Cross-Section Query Properties below). To determine the reference properties from the nose, body, and aft subobjects, *geometry-source-object* should have a value of *nil*. Directly entering values for the reference properties will override this behavior.

Cross-Section Query Properties:

The reference properties and some of the nose, body, and aft geometric properties can be determined directly from a vehicle fuselage geom. If the *geometry-source-object* property points to an object with a geom, the *cs-info* property will contain information describing the geometry of the fuselage at locations distributed along its length. The properties which control the determination of cross-section information are *cs-query-point*, *cs-query-vector*, *cs-query-coord-sys*, and *cs-query-cut-spacing*. The *cross-section-query* property is an instance of the class which computes the query results and should not be changed.

This object provides the entries for the fuselage (AXIBOD or ELLBOD) namelist in the Datcom input file. The actual Datcom input is written from an instance of *datcom-body-properties-class* which references an instance of *datcom-fuselage-interface-class*.

This class provides entries for the AXIBOD or ELLBOD namlist in the Datcom input file.

| Datcom Entry | Corresponding Property | Default Subobject | Source | Source Property | Subobject |
|--------------|---------------------------|-------------------|--------|----------------------|-----------|
| LNOSE | <i>datcom-input-lnose</i> | <i>nose-info</i> | | <i>nose-length</i> | |
| DNOSE | <i>datcom-input-dnose</i> | <i>nose-info</i> | | <i>nose-diameter</i> | |

| | | | |
|--------|----------------------------------|-------------------------|---------------------------|
| WNOSE | <i>datcom-input-dnose</i> | <i>nose-info</i> | <i>nose-diameter</i> |
| TNOSE | <i>datcom-input-tnose</i> | <i>nose-info</i> | <i>nose-type</i> |
| BNOSE | <i>datcom-input-bnose</i> | <i>nose-info</i> | <i>nose-bluntness</i> |
| TRUNC | <i>datcom-input-trunc</i> | <i>nose-info</i> | <i>nose-truncated?</i> |
| ENOSE | <i>datcom-input-enose</i> | <i>nose-info</i> | <i>nose-ellipticity</i> |
| POWER | <i>datcom-input-power</i> | <i>nose-info</i> | <i>nose-power</i> |
| | <i>datcom-input-nose-shape</i> | <i>nose-info</i> | <i>nose-cs-shape</i> |
| LCENTR | <i>datcom-input-lcentr</i> | <i>center-body-info</i> | <i>center-length</i> |
| DCENTR | <i>datcom-input-dcentr</i> | <i>center-body-info</i> | <i>center-diameter</i> |
| WCENTR | <i>datcom-input-dcentr</i> | <i>center-body-info</i> | <i>center-diameter</i> |
| ECENTR | <i>datcom-input-ecentr</i> | <i>center-body-info</i> | <i>center-ellipticity</i> |
| | <i>datcom-input-center-shape</i> | <i>center-body-info</i> | <i>center-cs-shape</i> |
| LAFT | <i>datcom-input-laft</i> | <i>aft-body-info</i> | <i>aft-length</i> |
| DAFT | <i>datcom-input-daft</i> | <i>aft-body-info</i> | <i>aft-diameter</i> |
| WAFT | <i>datcom-input-daft</i> | <i>aft-body-info</i> | <i>aft-diameter</i> |
| DEXIT | <i>datcom-input-dexit</i> | <i>aft-body-info</i> | <i>exit-diameter</i> |
| TAFT | <i>datcom-input-taft</i> | <i>aft-body-info</i> | <i>aft-type</i> |
| Eaft | <i>datcom-input-eaft</i> | <i>aft-body-info</i> | <i>aft-ellipticity</i> |
| | <i>datcom-input-aft-shape</i> | <i>aft-body-info</i> | <i>aft-cs-shape</i> |
| | <i>datcom-input-base-drag?</i> | <i>aft-body-info</i> | <i>ignore-base-drag?</i> |

| | | | |
|--|--------------|--|--|
| | <i>drag?</i> | | |
|--|--------------|--|--|

Inherit-From:

(datcom-input-source-object-mixin)

Properties:

- aero-coeffs-source-object** If this property points to an instance of *aerodynamic-coefficients-source-class*, then the aerodynamic coefficients provided by that object will be sent to Datcom instead of information describing the fuselage geometry. If the value of this property is *nil*, then the geometric properties will be written.
- geometry-source-object** This property should point to an object which has a geom representing the overall fuselage geometry or to an instance of *body-geometry-interface-class* which describes the fuselage. When it does, that geometry can be used to determine reference and cross-section properties.
- cs-query-point** The cross-section cuts will be made along the fuselage in the *cs-query-vector* direction starting at this point. The default value is '(0.0 0.0 0.0).
- cs-query-vector** This property contains the direction along which cross-section cuts will be made. The default value is '(1.0 0.0 0.0).
- cs-query-coord-sys** This property should point to an instance of *coordinate-system-class* in which the *cs-query-point* and *cs-query-vector* are defined. The default value of *nil* indicates the global coordinate system.
- cs-query-cut-spacing** This value of this property is the spacing between cross-section cuts that are made in the fuselage. The default value is 1.0.
- cross-section-query** This property is actually an instance of the class *cross-section-parameters-along-axis-class*. This object is used to determine the cross-section properties of the fuselage distributed along the length of the fuselage. This object should not be changed by the user.
- cs-info** This property contains the results of the cross-section query operation. The formula and value of this property should normally **not** be changed by the user.
- ref-info** This property contains the results of internal queries for the reference geometric properties of the fuselage (as described above). If the queries are not successful, the value of this property will be *nil*. If successful, it will contain a list of values: (*reference-area reference-diameter-latitude reference-diameter-longitude body-length*).
- reference-area** The reference cross-sectional area of the vehicle fuselage. By default, this will be the first value in the list contained in *ref-info*. If *ref-info* is *nil*, the default value of 1.0 will be used.
- reference-diameter-longitude** The longitudinal reference diameter of the vehicle fuselage. By default, this will be the third value in the list contained in *ref-info*. If *ref-info* is *nil*, the default value of 1.0 will be used.

| | |
|-----------------------------|--|
| reference-diameter-latitude | The lateral reference diameter of the vehicle fuselage. By default, this will be the second value in the list contained in <i>ref-info</i> . If <i>ref-info</i> is <i>nil</i> , the default value of 1.0 will be used. |
| body-length | The reference length of the vehicle fuselage. By default, this will be the fourth value in the list contained in <i>ref-info</i> . |

The default values of the following properties come from the values of properties in the subobjects of the *datcom-fuselage-interface-class*. The locations and names of these properties are given in the table above. A description of the properties can be found in the class definitions of their respective subobjects. The values of the subobject properties can be overridden by giving the *datcom-fuselage-interface-class* properties values directly.

datcom-input-lnose
datcom-input-dnose
datcom-input-lcentr
datcom-input-dcentr
datcom-input-laft
datcom-input-daft
datcom-input-dexit
datcom-input-base-drag?
datcom-input-tnose
datcom-input-bnose
datcom-input-trunc
datcom-input-enose
datcom-input-nose-shape
datcom-input-taft
datcom-input-eaft
datcom-input-aft-shape
datcom-input-ecentr
datcom-input-center-shape
datcom-input-power

datcom-wing-set-interface-class

[Class]

This object provides an interface between wing geometry and Datcom. If the *source-object* property points to an instance of *wing-set-class*, then the values of the other properties will be taken directly from that object (with the exception of *deflection-angles*). If *source-object* is *nil*, then all values must be set by the user.

This object provides the entries for the FINSET namelist in the Datcom input file. If the airfoil section type is NACA, it also provides information for the corresponding NACA entry in the input file.

| Datcom Entry | Corresponding Property | Datcom Entry | Corresponding Property |
|--------------|------------------------|--------------|------------------------------|
| SECTYP | <i>section-type</i> | ZUPPER | <i>wing-set-profile-info</i> |
| SSPAN | <i>semi-spans</i> | ZLOWER | <i>wing-set-profile-info</i> |
| CHORD | <i>chords</i> | LMAXU | <i>wing-set-profile-info</i> |
| XLE | <i>xle</i> | LMAXL | <i>wing-set-profile-info</i> |
| SWEEP | <i>sweep-angles</i> | LFLATU | <i>wing-set-profile-info</i> |
| STA | <i>sweep-stations</i> | LFALTL | <i>wing-set-profile-info</i> |
| NPANEL | <i>npanel</i> | LER | <i>wing-set-profile-info</i> |
| PHIF | <i>wing-phis</i> | CFOC | <i>flap-chord-ratios</i> |
| GAMMA | <i>wing-gammas</i> | | |

Inherit-From:

(datcom-input-source-object-mixin)

Properties:

| | |
|---------------|---|
| source-object | If this property points to an instance of <i>wing-set-class</i> , the values of the other properties in this object will be taken directly from that object (with the exception of <i>deflection-angles</i>). By default, this property has a value of <i>nil</i> and all values must be entered manually. |
| section-type | The section type of the airfoil used for the wing. Allowed values are <i>'hex</i> , <i>'naca</i> , <i>'arc</i> , and <i>'user</i> . The default value is <i>nil</i> . The object does not support passing <i>'user</i> -related information to Datcom. |
| semi-spans | A list of the span positions of each airfoil section. The default value in manual mode is '(0.0 1.0). |
| chords | A list of the chords of each airfoil section. The default value in manual mode is '(1.0 1.0). |
| xle | The position along the fuselage of the front of the root airfoil. The default value in manual mode is 0.0. |
| sweep-angles | The sweep angles between airfoils. The default value in manual mode is '(0.0). |

| | |
|-----------------------|---|
| sweep-stations | The relative position in each airfoil where sweep is measured. The default value in manual mode is '(0.0). |
| npanel | The number of wings in this wing set. The default value in manual mode is equal to the number of angles specified in <i>wing-phis</i> . |
| wing-phis | The angular positions (about the fuselage) of the wings in this wing set. The default value in manual mode is '(0.0). |
| wing-gammas | The root dihedral angles of the wings in this wing set. The default value in manual mode is '(0.0). |
| deflection-angles | The control-surface deflection angles of each wing in the wing set. The default value is <i>nil</i> . Even when <i>source-object</i> points to a <i>wing-set-class</i> , this value must be entered manually. |
| xhinge | The distance of the control surface hinge from the vehicle nose. The default value in manual mode is 0.0. |
| hinge-skew-angle | The location along the airfoil chord of the flap leading edge. The default value in manual mode is '(0.0 0.0). |
| flap-chord-ratios | |
| wing-set-profile-info | A list of information which describes the airfoil profiles. The contents of this list will depend on the airfoil type (<i>section-type</i>). For the <i>'hex</i> and <i>'arc</i> airfoil sections, different values may be entered for each airfoil along the span. <i>'hex</i> : (list <i>'hex</i> (list of <i>thickness-to-chord-ratio-upper</i>) (list of <i>thickness-to-chord-ratio-lower</i>) (list of <i>leading-edge-ratio-upper</i>) (list of <i>leading-edge-ratio-lower</i>) (list of <i>flat-ratio-upper</i>) (list of <i>flat-ratio-lower</i>) (list of <i>leading-edge-radius</i>)) <i>'arc</i> : (list <i>'arc</i> (list of <i>thickness-to-chord-ratio-upper</i>) (list of <i>thickness-to-chord-ratio-lower</i>) (list of <i>leading-edge-radius</i>)) <i>'naca</i> : (list <i>'naca</i> <i>naca-series-type</i> <i>naca-series-number</i>) |

datcom-protuberances-interface-class

[Class]

This object provides an interface between protuberance geometry and Datcom.

This object provides the entries for the PROTUB namelist in the Datcom input file.

| | | | |
|--------------|------------------------|--------------|------------------------|
| Datcom Entry | Corresponding Property | Datcom Entry | Corresponding Property |
|--------------|------------------------|--------------|------------------------|

| | | | |
|-------|--|-------|--|
| NPROT | | LPROT | |
| PTYPE | | WPROT | |
| XPROT | | HPROT | |
| XLOC | | OPROT | |

Inherit-From:

(datcom-input-source-object-mixin)

Properties:

| | |
|-----------------------|---|
| source-objects | This property should point to an instance of <i>protuberance-series-class</i> . The values written to the Datcom input file will be taken from this object. |
| num-protuberance-sets | The number of protuberance sets referred to in <i>source-objects</i> . This should not be changed by the user. |
| num-protuberances | The total number of protuberances in all protuberance sets. This should not be changed by the user. |

| | |
|------------------------------|---------|
| datcom-inlet-interface-class | [Class] |
|------------------------------|---------|

This object provides an interface between inlet geometry and Datcom. If the *inlet-geometry-object* property points to an instance of *inlet-set-class*, then the values of the other properties will be taken directly from that object. If *inlet-geometry-object* is *nil*, then all values must be set by the user.

This object provides the entries for the INLET namelist in the Datcom input file.

| Datcom Entry | Corresponding Property | Datcom Entry | Corresponding Property |
|--------------|----------------------------|--------------|------------------------|
| NIN | <i>datcom-input-nin</i> | X | <i>datcom-input-x</i> |
| INTYPE | <i>datcom-input-intype</i> | H | <i>datcom-input-h</i> |

| | | | |
|-------|---------------------------|-------|---------------------------|
| XINLT | <i>datcom-input-xinlt</i> | W | <i>datcom-input-w</i> |
| XDIV | <i>datcom-input-xdiv</i> | COVER | <i>datcom-input-cover</i> |
| HDIV | <i>datcom-input-hdiv</i> | RAMP | <i>datcom-input-ramp</i> |
| LDIV | <i>datcom-input-ldiv</i> | ADD | <i>datcom-input-add</i> |
| PHI | <i>datcom-input-phi</i> | MFR | <i>datcom-input-mfr</i> |

Inherit-From:

(datcom-input-source-object-mixin)

Properties:

| | |
|-----------------------|--|
| inlet-geometry-object | If this property points to an instance of <i>inlet-set-class</i> , the values of the other properties in this object will be taken directly from that object. By default, this property has a value of <i>nil</i> and all values must be set manually. |
| inlet-interface-data | If <i>inlet-geometry-object</i> points to an instance of <i>inlet-set-class</i> , then the inlet geometry properties in the class will be set automatically. |

The default values of the following properties are obtained directly from the object pointed to by *inlet-geometry-object*. The values may be set or overridden by giving the *datcom-inlet-interface-class* properties values directly.

datcom-input-nin
 datcom-input-intype
 datcom-input-xinlt
 datcom-input-xdiv
 datcom-input-hdiv
 datcom-input-ldiv
 datcom-input-phi
 datcom-input-x
 datcom-input-h
 datcom-input-w
 datcom-input-cover

datcom-input-ramp
 datcom-input-add
 datcom-input-mfr

datcom-nose-interface-class

[Class]

This class is normally used as a subobject of *datcom-fuselage-interface-class*. It acts as an interface between the vehicle nose geometry and the Datcom fuselage information. If all Datcom properties are being set manually, they can be changed in the *datcom-fuselage-interface-class* and this class may be ignored.

This class can operate in three different modes based on the value of the *source-mode* property: local, fuselage, and manual. The following table summarizes the input required for different values of *source-mode*. If the "user provides" and "can compute" requirements are not met, then all input will be manual.

| <i>source-mode</i> is | <i>'manual</i> | <i>'local</i> | | | <i>'fuselage</i> |
|--------------------------|----------------|-------------------------------|----------------|---------|-------------------------|
| user provides | nothing | <i>geometry-source-object</i> | | nothing | <i>fuselage-cs-info</i> |
| can compute | | <i>nose-source-info</i> | <i>cs-info</i> | | |
| <i>nose-length</i> | manual | automatic | computed | manual | computed |
| <i>nose-diameter</i> | manual | automatic | computed | manual | computed |
| <i>nose-type</i> | manual | automatic | manual | manual | manual |
| <i>nose-bluntness</i> | manual | automatic | manual | manual | manual |
| <i>nose-truncated?</i> | manual | automatic | manual | manual | manual |

| | | | | | |
|-------------------------|--------|-----------|--------|--------|--------|
| <i>nose-ellipticity</i> | manual | automatic | manual | manual | manual |
| <i>nose-cs-shape</i> | manual | automatic | manual | manual | manual |
| <i>nose-power</i> | manual | automatic | manual | manual | manual |

This class provides entries for the AXIBOD or ELLBOD namelist in the Datcom input file.

| Datcom Entry | Corresponding Property | Datcom Entry | Corresponding Property |
|--------------|------------------------|--------------|-------------------------|
| LNOSE | <i>nose-length</i> | TRUNC | <i>nose-truncated?</i> |
| DNOSE | <i>nose-diameter</i> | ENOSE | <i>nose-ellipticity</i> |
| TNOSE | <i>nose-type</i> | POWER | <i>nose-power</i> |
| BNOSE | <i>nose-bluntness</i> | | |

Inherit-From:

(datcom-input-source-object-mixin)

Properties:

- fuselage-geometry-source-object** This property should contain a *the*-reference to an instance of an object with a geom which represents the vehicle fuselage. This is available for reference but is not used by this object. Normally, the value of this property is available from the object's parent and its default formula *^geometry-source-object* should not be changed.
- fuselage-cs-info** This property contains information that describes cross-section information for the fuselage. Normally, the value of this property is available from the object's parent and its default formula *^cs-info* should not be changed.

| | |
|------------------------|---|
| source-mode | This property is used to specify where the values of the properties describing the nose geometry should come from. Allowed values are <i>'fuselage'</i> , <i>'local'</i> , and <i>'manual'</i> . |
| geometry-source-object | The object instance pointed to by this property is used differently depending on the value of <i>source-mode</i> . |
| nose-source-info | If the <i>source-mode</i> is <i>'local'</i> and the <i>geometry-source-object</i> points to a Datcom fuselage or nose geometry object, then this property will contain geometric information from that object. If these conditions are not met, then this property will have a value of <i>nil</i> . The formula of this property should not be changed by the user. |
| nose-info | This property will contain geometric information about the nose which depends on the values of other properties in this object. The formula of this property should not be changed by the user. |
| nose-length | The length of the vehicle nose. The default value depends on <i>source-mode</i> (see above). In manual mode the default value is 1.0 |
| nose-diameter | The diameter of the vehicle nose. The default value depends on <i>source-mode</i> (see above). In manual mode the default value is 1.128. |
| nose-type | The type of vehicle nose. The allowed values are <i>'cone'</i> , <i>'ogive'</i> , <i>'power'</i> , <i>'karman'</i> , and <i>'hack'</i> . In manual mode, the default value is <i>'cone'</i> . |
| nose-bluntness | The bluntness of the vehicle nose. The default value depends on <i>source-mode</i> (see above). In manual mode, the default value is 0.0. |
| nose-truncated? | Indicates whether the nose is truncated or not. Allowed values are <i>t</i> and <i>nil</i> . The default value depends on <i>source-mode</i> (see above). In manual mode the default value is <i>nil</i> . |
| nose-ellipticity | Indicates the ellipticity of the nose when <i>nose-cs-shape</i> is <i>'elliptical'</i> . The default value depends on <i>source-mode</i> (see above). In manual mode the default value is 1.0. |
| nose-cs-shape | Indicates whether the cross-sections of the nose are circular or elliptical. Allowed values are <i>'circular'</i> and <i>'elliptical'</i> . The default value depends on <i>source-mode</i> (see above). In manual mode the default value is <i>'circular'</i> . |
| nose-power | The power of the nose shape when <i>nose-type</i> is <i>'power'</i> . The default value depends on <i>source-mode</i> (see above). In manual mode the default value is 1.0. |
| cs-query-point | The cross-section cuts will be made along the nose geometry in the <i>cs-query-vector</i> direction starting at this point. The default value is <i>'(0.0 0.0 0.0)'</i> . This property is only required when the <i>source-mode</i> is <i>'local'</i> and <i>nose-source-info</i> is <i>nil</i> . |
| cs-query-vector | This property contains the direction along which cross-section cuts will be made. The default value is <i>'(1.0 0.0 0.0)'</i> . This property is only required when the <i>source-mode</i> is <i>'local'</i> and <i>nose-source-info</i> is <i>nil</i> . |
| cs-query-coord-sys | This property should point to an instance of <i>coordinate-system-class</i> in which the <i>cs-query-point</i> and <i>cs-query-vector</i> are defined. The default value of <i>nil</i> indicates the global coordinate system. This property is only required when the <i>source-mode</i> is <i>'local'</i> and <i>nose-source-info</i> is <i>nil</i> . |
| cs-query-cut-spacing | The value of this property is the spacing between the cross-section cuts that are made in the nose geometry. The default value is 1.0. This property is only required when the <i>source-mode</i> is <i>'local'</i> and <i>nose-source-info</i> is <i>nil</i> . |
| cross-section-query | This property is actually an instance of the class <i>cross-section-parameters-along-axis-class</i> . This object is used to determine the |

cs-info

cross-section properties of the nose geometry distributed along its length. This object should not be changed by the user.
This property contains the results of the cross-section query operation and is normally only demanded when the *source-mode* is *'local* and *nose-source-info* is *nil*. The formula and value of this property should normally **not** be changed by the user.

datcom-center-body-interface-class

[Class]

This class is normally used as a subobject of *datcom-fuselage-interface-class*. It acts as an interface between the vehicle center-body geometry and the Datcom fuselage information. If all Datcom properties are being set manually, they can be changed in the *datcom-fuselage-interface-class* and this class may be ignored.

This class can operate in three different modes based on the value of the *source-mode* property: local, fuselage, and manual. The following table summarizes the input required for different values of *source-mode*. If the "user provides" and "can compute" requirements are not met, then all input will be manual.

| <i>source-mode</i> is | <i>'manual</i> | <i>'local</i> | | | <i>'fuselage</i> |
|---------------------------|----------------|--------------------------------|----------------|--------|-------------------------|
| user provides | nothing | <i>geometry-source-object</i> | nothing | | <i>fuselage-cs-info</i> |
| can compute | | <i>center-body-source-info</i> | <i>cs-info</i> | | |
| <i>center-length</i> | manual | automatic | computed | manual | computed |
| <i>center-diameter</i> | manual | automatic | computed | manual | computed |
| <i>center-ellipticity</i> | manual | automatic | manual | manual | manual |
| <i>center-cs-shape</i> | manual | automatic | manual | manual | manual |

Note: Currently, a *source-mode* of *'fuselage* is not supported and it will work like *'manual*.

This class provides entries for the AXIBOD or ELLBOD namelist in the Datcom input file.

| Datcom Entry | Corresponding Property | Datcom Entry | Corresponding Property |
|--------------|------------------------|--------------|---------------------------|
| LCENTR | <i>center-length</i> | ECENTR | <i>center-ellipticity</i> |
| DCENTR | <i>center-diameter</i> | | |

Inherit-From:

(datcom-input-source-object-mixin)

Properties:

| | |
|--|--|
| <i>fuselage-geometry-source-object</i> | This property should contain a <i>the</i> -reference to an instance of an object with a geom which represents the vehicle fuselage. This is available for reference but is not used by this object. Normally, the value of this property is available from the object's parent and its default formula <i>^geometry-source-object</i> should not be changed. |
| <i>fuselage-cs-info</i> | This property contains information that describes cross-section information for the fuselage. Normally, the value of this property is available from the object's parent and its default formula <i>^cs-info</i> should not be changed. |
| <i>source-mode</i> | This property is used to specify where the values of the properties describing the nose geometry should come from. Allowed values are <i>'fuselage</i> , <i>'local</i> , and <i>'manual</i> . Currently, <i>'fuselage</i> is not supported and will behave like <i>'manual</i> . |
| <i>geometry-source-object</i> | The object instance pointed to by this property is used differently depending on the value of <i>source-mode</i> . |
| <i>center-length</i> | The length of the vehicle center body. The default value depends on <i>source-mode</i> (see above). In manual mode the default value is 1.0. |
| <i>center-diameter</i> | The diameter of the vehicle center body. The default value depends on <i>source-mode</i> (see above). In manual mode the default value is 1.128. |

| | |
|----------------------|--|
| center-ellipticity | Indicates the ellipticity of the center body when <i>center-cs-shape</i> is 'elliptical'. The default value depends on <i>source-mode</i> (see above). In manual mode the default value is 1.0 |
| center-cs-shape | Indicates whether the cross-sections of the nose are circular or elliptical. Allowed values are 'circular' and 'elliptical'. The default value depends on <i>source-mode</i> (see above). In manual mode the default value is 'circular'. |
| cs-query-point | The cross-section cuts will be made along the center body geometry in the <i>cs-query-vector</i> direction starting at this point. The default value is '(0.0 0.0 0.0)'. This property is only required when the <i>source-mode</i> is 'local' and <i>nose-source-info</i> is <i>nil</i> . |
| cs-query-vector | This property contains the direction along which cross-section cuts will be mad. The default value is '(1.0 0.0 0.0)'. This property is only required when the <i>source-mode</i> is 'local' and <i>nose-source-info</i> is <i>nil</i> . |
| cs-query-coord-sys | This property should point to an instance of <i>coordinate-system-class</i> in which the <i>cs-query-point</i> and <i>cs-query-vector</i> are defined. The default value of <i>nil</i> indicates the global coordinate system. This property is only required when the <i>source-mode</i> is 'local' and <i>nose-source-info</i> is <i>nil</i> . |
| cs-query-cut-spacing | The value of this property is the spacing between the cross-section cuts that are made in the center body geometry. The default value is 1.0. This property is only required when the <i>source-mode</i> is 'local' and <i>nose-source-info</i> is <i>nil</i> . |
| cross-section-query | This property is actually an instance of the class <i>cross-section-parameters-along-axis-class</i> . This object is used to determine the cross-section properties of the center body geometry distributed along its length. This object should not be changed by the user. |
| cs-info | This property contains the results of the cross-section query operation and is normally only demanded when the <i>source-mode</i> is 'local' and <i>center-body-source-info</i> is <i>nil</i> . The formula and value of this property should normally not be changed by the user. |

datcom-aft-body-interface-class

[Class]

This class is normally used as a subobject of *datcom-fuselage-interface-class*. It acts as an interface between the vehicle aft body geometry and the Datcom fuselage information. If all Datcom properties are being set manually, they can be changed in the *datcom-fuselage-interface-class* and this class may be ignored.

This class can operate in three different modes based on the value of the *source-mode* property: local, fuselage, and manual. The following table summarizes the input required for different values of *source-mode*. If the "user provides" and "can compute" requirements are not met, then all input will be manual.

| <i>source-mode</i> is | <i>'manual</i> | <i>'local</i> | | | <i>'fuselage</i> |
|--------------------------|----------------|-------------------------------|----------------|--------|-------------------------|
| user provides | nothing | <i>geometry-source-object</i> | nothing | | <i>fuselage-cs-info</i> |
| can compute | | <i>aft-body-source-info</i> | <i>cs-info</i> | | |
| <i>aft-length</i> | manual | automatic | computed | manual | computed |
| <i>aft-diameter</i> | manual | automatic | computed | manual | computed |
| <i>forward-diameter</i> | manual | automatic | computed | manual | computed |
| <i>exit-diameter</i> | manual | automatic | | | |
| <i>ignore-base-drag?</i> | manual | manual | manual | manual | manual |
| <i>aft-type</i> | manual | automatic | manual | manual | manual |
| <i>aft-ellipticity</i> | manual | automatic | manual | manual | manual |
| <i>aft-cs-shape</i> | manual | automatic | manual | manual | manual |

Note: Currently, a *source-mode* of *'fuselage* is not supported and it will work like *'manual*.

This class provides entries for the AXIBOD or ELLBOD namelist in the Datcom input file.

| Datcom Entry | Corresponding Property | Datcom Entry | Corresponding Property |
|--------------|------------------------|--------------|------------------------|
| LAFT | <i>aft-length</i> | TAFT | <i>aft-type</i> |

| | | | |
|-------|----------------------|------|------------------------|
| DAFT | <i>aft-diameter</i> | EAFT | <i>aft-ellipticity</i> |
| DEXIT | <i>exit-diameter</i> | | |

Inherit-From:

(datcom-input-source-object-mixin)

Properties:

| | |
|---------------------------------|--|
| fuselage-geometry-source-object | This property should contain a <i>the</i> -reference to an instance of an object with a geom which represents the vehicle fuselage. This is available for reference but is not used by this object. Normally, the value of this property is available from the object's parent and its default formula <i>^geometry-source-object</i> should not be changed. |
| fuselage-cs-info | This property contains information that describes cross-section information for the fuselage. Normally, the value of this property is available from the object's parent and its default formula <i>^cs-info</i> should not be changed. |
| source-mode | This property is used to specify where the values of the properties describing the aft body geometry should come from. Allowed values are <i>'fuselage'</i> , <i>'local'</i> , and <i>'manual'</i> . Currently, <i>'fuselage'</i> is not supported and will behave like <i>'manual'</i> . |
| geometry-source-object | The object instance pointed to by this property is used differently depending on the value of <i>source-mode</i> . |
| aft-length | The length of the vehicle aft body. The default value depends on <i>source-mode</i> (see above). In manual mode the default value is 1.0. |
| aft-diameter | The diameter of the aft end of the vehicle aft body. The default value depends on <i>source-mode</i> (see above). In manual mode the default value is 1.128. |
| forward-diameter | The diameter of the front of the vehicle aft body. The default value depends on <i>source-mode</i> (see above). |
| exit-diameter | |
| ignore-base-drag? | |
| aft-type | The type of vehicle aft body. The allowed values are <i>'cone'</i> , <i>'conical'</i> , and <i>'ogive'</i> . In manual mode the default value is <i>'conical'</i> . |
| aft-ellipticity | Indicates the ellipticity of the nose when <i>aft-cs-shape</i> is <i>'elliptical'</i> . The default value depends on <i>source-mode</i> (see above). In manual mode the default value is 1.0. |
| aft-cs-shape | Indicates whether the cross-sections of the aft body are circular or elliptical. Allowed values are <i>'circular'</i> and <i>'elliptical'</i> . The default value depends on <i>source-mode</i> (see above). In manual mode the default value is <i>'circular'</i> . |

| | |
|----------------------|--|
| cs-query-point | The cross-section cuts will be made along the aft body geometry in the <i>cs-query-vector</i> direction starting at this point. The default value is '(0.0 0.0 0.0). This property is only required when the <i>source-mode</i> is 'local and <i>nose-source-info</i> is nil. |
| cs-query-vector | This property contains the direction along which cross-section cuts will be mad. The default value is '(1.0 0.0 0.0). This property is only required when the <i>source-mode</i> is 'local and <i>nose-source-info</i> is nil. |
| cs-query-coord-sys | This property should point to an instance of <i>coordinate-system-class</i> in which the <i>cs-query-point</i> and <i>cs-query-vector</i> are defined. The default value of nil indicates the global coordinate system. This property is only required when the <i>source-mode</i> is 'local and <i>nose-source-info</i> is nil. |
| cs-query-cut-spacing | The value of this property is the spacing between the cross-section cuts that are made in the aft body geometry. The default value is 1.0. This property is only required when the <i>source-mode</i> is 'local and <i>nose-source-info</i> is nil. |
| cross-section-query | This property is actually an instance of the class <i>cross-section-parameters-along-axis-class</i> . This object is used to determine the cross-section properties of the aft body geometry distributed along its length. This object should not be changed by the user. |
| cs-info | This property contains the results of the cross-section query operation and is normally only demanded when the <i>source-mode</i> is 'local and <i>aft-body-source-info</i> is nil. The formula and value of this property should normally not be changed by the user. |

datcom-input-source-object-mixin

[Class]

This class exists to provide a common class for the interface classes to inherit from. It adds no special properties or behavior.

Inherit-From:

(object)

Properties:

Appendix 4: AML Panair Interface Documentation

This documentation assumes that the user is familiar with the parameters required for the PANAIR input file.

The *basic-panair-class* provides the primary interface to the PANAIR application. Control of PANAIR is through properties in this object and in its children. Additional subobjects will need to be added (particularly to the *networks* subobject) to interface to the vehicle geometry.

All of the capabilities of this interface to PANAIR can be accessed through properties. Users will normally not need to call any methods.

The *basic-panair-class* object will compute results for multiple values of angle of attack, sideslip angle, and Mach numbers by performing multiple runs of PANAIR. Typically, the ability to run multiple solutions of during one run of PANAIR, although available, is not used. This functionality is discussed under the *panair-onset-flow-conditions-properties-class* documentation.

Two classes are provided. The *basic-panair-class* provides the basic interface to PANAIR. The *panair-class* inherits from this class and adds the common aerodynamic coefficient access interface provided by *aerodynamic-coefficients-source-class*.

| | |
|---------------------------|---------|
| basic-panair-class | [Class] |
|---------------------------|---------|

This class provides the primary interface to the PANAIR application.

The *basic-panair-class* has several subobjects, most of which have properties that should be changed by the user. In order to change the values or formulas of these properties, new classes will need to be created (inheriting from the originals) with replacement

formulas and put in the subobjects section of a new class inheriting from *basic-panair-class*. The subobjects, their classes, and whether they contain user-editable properties are summarized in this table:

| Subobject Name | Subobject Class | User Editable? |
|---|--|----------------|
| title-properties | panair-title-properties-class | yes |
| run-type-properties | panair-run-type-properties-class | yes |
| flow-symmetry-properties | panair-flow-symmetry-properties-class | yes |
| onset-flow-conditions-properties | panair-onset-flow-conditions-properties-class | yes |
| reference-data-properties | panair-reference-data-properties-class | yes |
| networks | panair-networks-class | no |
| printout-control-properties | panair-printout-control-properties-class | yes |
| boundary-layer-and-velocity-correction-control-properties | panair-boundary-layer-and-velocity-correction-control-properties-class | yes |
| liberalized-abutments-properties | panair-liberalized-abutments-properties-class | yes |

The flight conditions for each run of PANAIR will be determined from the properties *mach-numbers*, *angles-of-attack*, and *sideslip-angles*. A separate run of PANAIR will be made for each combination of the values in these properties. To make only a single PANAIR run, each of these properties should contain a list with a single value.

PANAIR has an option to find multiple solutions during a single run for angles of attack and sideslip angles close to the compressibility direction angles. This mode is normally not used by *basic-panair-class*. To make a single PANAIR run with multiple solutions, at least one of the properties *mach-numbers*, *angles-of-attack*, and *sideslip-angles* should

have a value of *nil*. The flight conditions will then be written to the input file based on the properties in the *panair-onset-flow-conditions-properties-class* subobject.

The results of the PANAIR runs are contained in the property *component-coefficients*. The form of the results is a list of lists of data. In each list, the first list contains the flight conditions and the second list contains the resulting coefficients. Each entry in the list has the form:

((mach-number angle-of-attack sideslip-angle) (fx fy fz mx my mz area)).

The data is read from the PANAIR *ffmf* output file after each run.

Two logical paths are expected to be available when using the *basic-panair-class*. The first, *:panair-path*, should point to the location of the PANAIR executable. The second, *:panair-user-data*, should point to the parent directory where the data files for PANAIR runs should be stored. This class will automatically create subdirectories under the path specified by *:panair-user-data*.

Inherit-from:

(name-generator)

Properties:

| | |
|------------------------|---|
| mach-numbers | A list of values to be used as the Mach number input for a PANAIR run. |
| angles-of-attack | A list of values to be used as the angle of attack input for a PANAIR run. |
| sideslip-angles | A list of values to be used as the sideslip angle input for a PANAIR run. |
| body-length | A reference vehicle length. This will be used in the automatic computation of values in the <i>panair-reference-data-properties-class</i> object. The default value is 1.0. |
| max-cross-section-area | A reference vehicle area. This will be used in the automatic computation of values in the <i>panair-reference-data-properties-class</i> object. The default value is 1.0. |
| component-coefficients | A list containing the results of all of the PANAIR runs made. This should not be changed by the user. |

| | |
|-------------------------|--|
| panair-model-name | This name will be written to the PANAIR input file and will be used as the name of the directory in which data files are stored. The default formula will append the date to the string "PANAIR-MODEL". |
| panair-input-file-name | The complete path and filename for the input file to PANAIR. The default value is the <i>panair-model-name</i> with the extension ".inp" in the directory specified by <i>run-dir</i> . This is the file that is written by <i>basic-panair-class</i> before PANAIR is run. |
| panair-output-file-name | The complete path and filename for the main output file from PANAIR. The default value is the <i>panair-model-name</i> with the extension ".out" in the directory specified by <i>run-dir</i> . Note: This is only used when running on a UNIX system. On Windows systems, the output file name will be "panair.out". |
| panair-command | This property should contain a string containing the complete command required to run PANAIR. |
| run-panair? | Demanding this property will cause PANAIR to be run. The results from the run will be stored in this property and <i>component-coefficients</i> . They should be read from <i>component-coefficients</i> . |
| run-dir | The property contains the complete path to the directory in which PANAIR files will be written. By default, this property will get its value from the <i>directory</i> property and automatically create the directory if it does not already exist. This property should be treated as an output and should not be changed by the user. |
| directory | This property should contain the complete path to the directory in which PANAIR files will be written. The default will be a directory with the name <i>panair-model-name</i> in the directory specified by the <i>:panair-user-data</i> logical path. This property should be treated as an input . The actual directory being used should be read from the <i>run-dir</i> property. Accessing that property will automatically create the directory if it does not already exist. |

Methods:

write-panair-properties

write-panair-deck

panair-title-properties-class

[Class]

This class is used as a subobject of *basic-panair-class* and provides information for the TITLE data block of the PANAIR input file. The strings contained in *panair-model-name* and *title-lines* will be put in this block.

Inherit-from:

(panair-properties-class)

Properties:

| | |
|-------------------|--|
| panair-model-name | A string which will be put as the first line in the TITLE data block. The default formula will use the value of the <i>panair-model-name</i> of the parent <i>basic-panair-class</i> . |
| title-lines | A list of string which will be put, in order, in the TITLE data block following <i>panair-model-name</i> . The default value is <i>nil</i> . |

Methods:

write-panair-properties

| | |
|----------------------------------|---------|
| panair-run-type-properties-class | [Class] |
|----------------------------------|---------|

This class is used as a subobject of *basic-panair-class* and provides information for the SOLUTION, DATACHECK, and RESTART data blocks of the PANAIR input file. The type of data block written will be based on the value of the *program-control-mode* property.

Inherit-from:

(panair-properties-class)

Properties:

| | |
|----------------------|---|
| program-control-mode | The value of this property determines the mode in which PANAIR will be run. This will determine which of the other properties will be written into the PANAIR input file. Allowed values are ' <i>solution</i> ', ' <i>datacheck</i> ', and ' <i>restart</i> '. The default value is ' <i>solution</i> '. |
| ndtchk | The NDTCHK parameter in the PANAIR input file. This parameter is written when <i>program-control-mode</i> is ' <i>datacheck</i> '. The default value is 1.0. |
| nckusp | The NCKUSP parameter in the PANAIR input file. This parameter is written when <i>program-control-mode</i> is ' <i>restart</i> '. The default value is 0.0. |

| | |
|--------|--|
| nckaic | The NCKAIC parameter in the PANAIR input file. This parameter is written when <i>program-control-mode</i> is 'restart'. The default value is 1.0. |
| nckfam | The NCKFAM parameter in the PANAIR input file. This parameter is written when <i>program-control-mode</i> is 'restart'. The default value is 1.0 if <i>nckaic</i> = 1.0 and 0.0 otherwise. |

Methods:

write-panair-properties

panair-flow-symmetry-properties-class [Class]

This class is used as a subobject of *basic-panair-class* and provides information for the SYMMETRY data block of the PANAIR input file.

Inherit-from:

(panair-properties-class)

Properties:

| | |
|-------|---|
| xzpln | The XZPLN parameter in the PANAIR input file. The default value is 0.0. |
| xypln | The XYPLN parameter in the PANAIR input file. The default value is 0.0. |

Methods:

write-panair-properties

panair-onset-flow-conditions-properties-class [Class]

This class is used as a subobject of *basic-panair-class* and provides information for the MACH, CASES, ANGLES, and YAW data blocks of the PANAIR input file. The properties in this object are only used when the user does **not** specify values for the

properties *mach-numbers*, *angles-of-attack*, and *sideslip-angles*. If all three of these properties are specified, then the data blocks will be written based on their values and the properties of this object will be ignored. If they are not all specified, then the data blocks will be written based on the properties of this object.

The only way a single PANAIR run may be made with more than one angle of attack or sideslip angle is by using the *alpha* and *beta* properties of this object.

If the properties *mach-numbers*, *angles-of-attack*, and *sideslip-angles* are given values in the *basic-panair-class* object then any values given to properties of this object will be ignored.

Inherit-from:

(panair-properties-class)

Properties:

| | |
|--------|--|
| amach | The Mach number at which the simulation will be run. This is the AMACH parameter in the PANAIR input file. |
| nacase | The number of solution cases in the PANAIR run. This is the NACASE parameter in the PANAIR input file. The default formula is the minimum length of the <i>alpha</i> and <i>beta</i> property lists. |
| alpc | The angle of attack direction of compressibility effects. This is the ALPC parameter in the PANAIR input file. |
| alpha | A list of up to four (4) angles of attack which will be used to compute multiple solutions during a single PANAIR run. This is the ALPHA() parameter in the PANAIR input file. The default formula is a list with a single entry equal to the value of <i>alpc</i> . |
| betc | The sideslip angle direction of compressibility effects. |
| beta | A list of up to four (4) sideslip angles which will be used to compute multiple solutions during a single PANAIR run. This is the BETA() parameter in the PANAIR input file. The default formula is a list with a single entry equal to the value of <i>betc</i> . |

Methods:

write-panair-properties

This class is used as a subobject of *basic-panair-class* and provides information for the REFERENCE data block of the PANAIR input file.

Inherit-from:

(panair-properties-class)

Properties:

| | |
|-------------|--|
| body-length | The length of the body. The default formula will get this value from the <i>body-length</i> property of the parent <i>basic-panair-class</i> and should not be changed. This property is used directly by PANAIR. |
| xref | The x component of the moment reference location corresponding to the XREF parameter in the PANAIR input file. The default value is 52% of the <i>body-length</i> . |
| yref | The y component of the moment reference location corresponding to the YREF parameter in the PANAIR input file. The default value is 0.0. |
| zref | The z component of the moment reference location corresponding to the ZREF parameter in the PANAIR input file. The default value is 0.0. |
| sref | The full airplane reference area corresponding to the SREF parameter in the PANAIR input file. The default formula will get this value from the <i>max-cross-section-area</i> property of the parent <i>basic-panair-class</i> . |
| bref | The reference length for MX corresponding to the BREF parameter in the PANAIR input file. The default formula will compute the radius of a circle with area equal to <i>sref</i> . |
| cref | The reference length for MY corresponding to the CREF parameter in the PANAIR input file. The default formula will use the value of <i>bref</i> . |
| dref | The reference length for MZ corresponding to the DREF parameter in the PANAIR input file. The default formula will use the value of <i>bref</i> . |

Methods:

write-panair-properties

This class is used as a subobject of *basic-panair-class* and provides information for the PRINTOUT data block of the PANAIR input file.

Inherit-from:

(panair-properties-class)

Properties:

| | |
|--------|---|
| isings | Corresponds to the ISINGS parameter in the PANAIR input file. The default value is 0.0. |
| igeomp | Corresponds to the IGEOMP parameter in the PANAIR input file. The default value is 0.0. |
| isingp | Corresponds to the ISINGP parameter in the PANAIR input file. The default value is 0.0. |
| icontp | Corresponds to the ICONTP parameter in the PANAIR input file. The default value is 0.0. |
| ibconp | Corresponds to the IBCONP parameter in the PANAIR input file. The default value is 0.0. |
| iedgep | Corresponds to the IEDGEP parameter in the PANAIR input file. The default value is 0.0. |
| ipraic | Corresponds to the IPRAIC parameter in the PANAIR input file. The default value is 0.0. |
| nexdgn | Corresponds to the NEXDGN parameter in the PANAIR input file. The default value is 0.0. |
| ioutpr | Corresponds to the IOPTR parameter in the PANAIR input file. The default value is 1.0. |
| ifmcpr | Corresponds to the IFMCPR parameter in the PANAIR input file. The default value is 0.0. |
| icostp | Corresponds to the ICOSTP parameter in the PANAIR input file. The default value is 0.0. |

Methods:

write-panair-properties

This class is used as a subobject of *basic-panair-class* and provides information for the

BOUNDARY and VELOCITY data blocks of the PANAIR input file. The type of data block written will be based on the value of the *bl-or-ivcorr?* property.

Inherit-from:

(panair-properties-class)

Properties:

| | |
|----------------------------|--|
| bl-or-ivcorr? | This property determines whether the BOUNDARY or VELOCITY data block will be written to the PANAIR input file. Allowed values are <i>'boundary'</i> and <i>'velocity'</i> . The default value is <i>'boundary'</i> . |
| ivcorr-boundary-layer | The value of this property will be used as the IVCORR parameter in the PANAIR input file when <i>bl-or-ivcorr?</i> has a value of <i>'boundary'</i> . The default value is 0.0. |
| ivcorr-velocity-correction | The value of this property will be used as the IVCORR parameter in the PANAIR input file when <i>bl-or-ivcorr?</i> has a value of <i>'velocity'</i> . The default value is 0.0. |

Methods:

write-panair-properties

panair-liberalized-abutments-properties-class

[Class]

This class is used as a subobject of *basic-panair-class* and provides information for the EAT data block of the PANAIR input file. The property *liberalized-abutments?* is used to indicate whether this data block should be written to the file or not.

Inherit-from:

(panair-properties-class)

Properties:

| | |
|------------------------|---|
| liberalized-abutments? | When this property has a value of <i>t</i> , the EAT data block will be written to the PANAIR input file. The default value is <i>nil</i> . |
| epsgeo | The EPSGEO parameter in the PANAIR input file. The default value is 0.1. |
| igeoin | The IGEOIN parameter in the PANAIR input file. The default value of <i>nil</i> will create a blank entry in the file. |
| igeout | The IGEOOUT parameter in the PANAIR input file. The default value of <i>nil</i> will create a blank entry in the file. |
| nwxref | The NWXREF parameter in the PANAIR input file. The default value of <i>nil</i> will create a blank entry in the file. |
| triint | The TRIINT parameter in the PANAIR input file. The default value of <i>nil</i> will create a blank entry in the file. |
| iabsum | The IABSUM parameter in the PANAIR input file. The default value is 0.0. |

Methods:

write-panair-properties

panair-networks-class

[Class]

This class is used as a subobject of *basic-panair-class* and provides information for the POINTS data blocks of the PANAIR input file.

The *network-list* property should contain a list of references to instances of class *panair-network-class*. By default, the formula for this property will find all instances of *panair-network-class* that are in the branch of the model tree headed by *panair-networks-class*. Any networks that are added beneath this object will automatically be added to the list. Alternately, networks may be placed anywhere in the tree and the formula for *network-list* may be changed appropriately.

Inherit-from:

(panair-properties-class)

Properties:

| | |
|----------------------|--|
| network-list | This property should contain a list of references to instances of <i>panair-network-class</i> which represent the actual networks to be used in the analysis. The default formula will find all instances of <i>panair-network-class</i> which are under the <i>panair-networks-class</i> . If this default behavior is acceptable, the formula for this property should not be changed. |
| network-lookup-table | This property contains a list associating instances of <i>panair-network-class</i> and their network ids. This property should not be changed by the user. |
| network-ids | This property contains a list of network ids corresponding to the objects in <i>network-list</i> . The individual network objects will get their ids from this property (through the method <i>get-network-id</i>). By default, this property will contain a list of consecutive integers beginning with 1. This property should not normally be changed by the user. |

Methods:

write-panair-properties

get-network-id

panair-network-class

[Class]

This class is used to represent one network in the PANAIR analysis. Information in this object will be used to create a POINTS data block in the PANAIR input file.

Typically, instances of this class will be children of an instance of *panair-networks-class* but they may appear anywhere in the model tree. If they do not appear beneath the *panair-networks-class* instance, then the *network-list* of the *panair-networks-class* instance will need to be changed to reflect this.

The *source-object* property should point to an instance of an object from which mesh geometry can be found.

Inherit-from:

(web-surface-from-points-object panair-properties-class)

Properties:

| | |
|-----------------|---|
| source-object | This property should point to an instance of an object which can provide network mesh information. The default value is <i>nil</i> . |
| networks-object | This property should point to an instance of <i>panair-networks-class</i> . By default, it will look up in the tree from the <i>panair-network-class</i> to find an instance of <i>panair-networks-class</i> . |
| network-id | This property contains the id for this network. By default, this will be found by calling the method <i>get-network-id</i> on the instance of <i>panair-networks-class</i> referenced in <i>networks-object</i> . This should normally not be changed by the user. |
| run-dir | The directory in which files should be written. By default, this property will get its value from a property with the same name in and instance above it in the tree. |
| netname | By default, this will be a string version of the <i>network-id</i> . This name is written as NETNAME to the PANAIR input file. |
| cpnorm | Describes the mesh surface normals and corresponds to the CPNORM parameter in the PANAIR input file. The default value is 2.0. |
| dnsmsh | Determines whether panel mesh density will be modified from that given by <i>nm</i> and <i>nn</i> and corresponds to the DNSMSH parameter in the PANAIR input file. When <i>dnsmsh</i> is 1.0, the parameters <i>dn</i> and <i>dm</i> will be written to the input file. When <i>dnsmsh</i> is 0.0, they will not. The default value is 0.0. |
| dn | Density factor which changes the number of columns in the mesh corresponding to the DN parameter in the PANAIR input file. This parameter will only be written to the file when <i>dnsmsh</i> is 1.0. The default value is 1.0. |
| dm | Density factor which changes the number of rows in the mesh corresponding to the DN parameter in the PANAIR input file. This parameter will only be written to the file when <i>dnsmsh</i> is 1.0. The default value is 1.0. |
| betl | Two-dimensional mass flux for each solution corresponding to BETL in the PANAIR input file. The value of this property should be a list of values corresponding to the cases in the <i>panair-onset-flow-conditions-property-class</i> . This property will be used to create the INFLOW data block in the input file and is only written when <i>kt</i> is either 9, 4, or 14. |
| betl-t | Twice the surface thickness slope for each solution corresponding to BETL in the PANAIR input file. The value of this property should be a list of values corresponding to the cases in the <i>panair-onset-flow-conditions-property-class</i> . This property will be used to create the SIMULATED THICKENSS data block in the input file and is only written when <i>kt</i> is either 2 or 12 and when <i>tkw</i> is 1.0. |
| kn | The number of networks in the group represented by this object corresponding to the KN parameter in the PANAIR input file. The default value is 1.0 and should not be changed by the user. |
| nm | The number of rows in the network corresponding to the NM parameter in the PANAIR input file. The value of this property is determined automatically based on the value of <i>points-list</i> and should not be changed by the user. |
| nn | The number of columns in the network corresponding to the NN parameter in the PANAIR input file. The value of this property is determined automatically based on the value of <i>points-list</i> and should not be changed by the user. |

| | |
|----------------------------|---|
| n-panels | The total number of panels in the network. This is computed automatically based on <i>nm</i> and <i>nn</i> and should not be changed by the user. |
| kt | Defines the boundary condition type for the mesh and corresponds to the KT parameter in the PANAIR input file. The default value is 1.0. |
| tkw | Corresponds to the TKW parameter in the PANAIR input file. The default value of <i>nil</i> will leave this blank in the file. |
| resultant-number-of-panels | The total number of panels in the network including the effects of the density properties <i>dn</i> and <i>dm</i> . This is computed automatically and should not be changed by the user. |
| points-list | A list of lists of points specifying the mesh for this network. When <i>source-object</i> is given, these points will be determined from the specified object and this property should not be changed by the user. These points will be written as X Y Z data in the PANAIR input file. |
| nodes-file | This property is used to create the network geometry if required (primarily for display purposes). It comes from the inheritance from <i>web-surface-from-points-object</i> . The default formula will create a file in the <i>run-dir</i> directory. It does not normally need to be changed by the user. |
| con-file | This property is used to create the network geometry if required (primarily for display purposes). It comes from the inheritance from <i>web-surface-from-points-object</i> . The default formula will create a file in the <i>run-dir</i> directory. It does not normally need to be changed by the user. |
| use-files? | This property is used to create the network geometry if required (primarily for display purposes). It comes from the inheritance from <i>web-surface-from-points-object</i> . It does not normally need to be changed by the user. |
| cleanup? | This property is used to create the network geometry if required (primarily for display purposes). It comes from the inheritance from <i>web-surface-from-points-object</i> . It does not normally need to be changed by the user. |

Methods:

generate-panair-network-mesh-points

write-panair-properties

panair-properties-class

[Class]

This class provides a common superclass for all PANAIR classes to inherit from. It adds no additional functionality.

Inherit-from:

(object)

PANAIR Class with Common Aerodynamic Coefficients Interface

panair-class

[Class]

This class combines the PANAIR analysis of *basic-panair-class* with the common interface for accessing aerodynamic coefficients provided by *aerodynamic-coefficients-source-class*. This will allow any other objects or applications which require the aerodynamic coefficients results from PANAIR to access them in a standard way. These can be accessed using the *get-coefficients* method and the key and coefficient symbols listed below.

This class also implements interfaces to flight condition and body geometry objects. If these objects are referenced, then many of the properties of *panair-class* will be computed automatically. The *panair-class* will ignore the state of the *angle-mode* property of the *flight-conditions-object* and always uses the *angles-of-attack* and *sideslip-angles* from the *flight-conditions-object*.

The aerodynamic coefficients are accessed using the keys 'M', 'alpha', and 'beta' summarized in this table:

| Key | Symbol | Label | Description | Range in Property: |
|-----------------|--------|---------|-------------------|-------------------------|
| Mach number | 'M | "M" | "Mach Number" | <i>mach-numbers</i> |
| angle of attack | 'alpha | "alpha" | "Angle of Attack" | <i>angles-of-attack</i> |
| sideslip angle | 'beta | "beta" | "Sideslip Angle" | <i>sideslip-angles</i> |

The aerodynamic coefficients available in *panair-class* are summarized in this table:

| Coefficient Symbol | PANAIR Value | Output |
|-----------------------|-----------------|--------|
| 'CA | fx | |
| 'CY | fy | |
| 'CN | fz | |
| 'CSL | mx | |
| 'CSM | my | |
| 'CSN | mz | |

Inherit-from:

(basic-panair-class aerodynamic-coefficients-source-class)

Properties:

- flight-conditions-object** If this property contains a *the*-reference to an instance of *flight-conditions-class*, then the values of *mach-numbers*, *angles-of-attack*, and *sideslip-angles* will be taken from that object and do not have to be set in this object. The default value of *nil* indicates that the properties need to be set in this object.
- body-geometry-interface-object** If this property contains a *the*-reference to an instance of *body-geometry-interface-class*, then the values of *body-length* and *max-cross-section-area* will be taken from that object and do not have to be set in this object. The default value of *nil* indicates that the properties need to be set in this object.
- body-length** The reference vehicle length. If a *body-geometry-interface-object* is specified, then the value of this property will be taken from the *body-length* property of that object. Otherwise, it must be set here.

| | |
|------------------------|---|
| max-cross-section-area | The reference vehicle area. If a <i>body-geometry-interface-object</i> is specified, then the value of this property will be taken from the <i>body-max-cs-area</i> of that object. Otherwise, it must be set here. |
| mach-numbers | A list of values to be used as the Mach number input for a PANAIR run. If a <i>flight-conditions-object</i> is specified, then the value of this property will be taken from the <i>mach-numbers</i> property of this object. Otherwise, it must be set here. |
| angles-of-attack | A list of values to be used as the angle of attack input for a PANAIR run. If a <i>flight-conditions-object</i> is specified, then the value of this property will be taken from the <i>angles-of-attack</i> property of this object. Otherwise, it must be set here. |
| sideslip-angles | A list of values to be used as the sideslip angle input for a PANAIR run. If a <i>flight-conditions-object</i> is specified, then the value of this property will be taken from the <i>sideslip-angles</i> property of this object. Otherwise, it must be set here. |
| ac-key-symbols | A list of the keys used to access the aerodynamic coefficients in the <i>panair-class</i> . This should not be changed by the user. |
| ac-key-labels | A list of the labels of the keys in <i>panair-class</i> . This should not be changed by the user. |
| ac-key-descriptions | A list of the descriptions of the keys in <i>panair-class</i> . This should not be changed by the user. |
| ac-key-ranges | A list of the key ranges in <i>panair-class</i> . This is based on the <i>mach-numbers</i> , <i>angles-of-attack</i> , and <i>sideslip-angles</i> properties and should not be changed by the user. |
| ac-coefficient-symbols | A list of the coefficient symbols in <i>panair-class</i> . This should not be changed by the user. |

Appendix 5: AML Dracon Interface Documentation

generate-drakon-materials-file

[Function]

This function generates the materials file needed for the Dracon analysis.

Format:

(generate-drakon-materials-file file-name &key drakon-path)

Arguments:

file-name String specifying the materials file name

drakon-path String specifying the path to the drakon directory. Default is (*logical-path*
:drakon-binaries)

convert-aml-mesh-ascii-files-to-drakon-binary

[Function]

This function converts the AML mesh ASCII files to drakon binary format for the Dracon analysis.

Format:

(convert-aml-mesh-ascii-files-to-drakon-binary

crd-file n-nodes con-file n-elements directory element-type

&key drakon-path)

Arguments:

crd-file String specifying the node coordinates file name

n-nodes Number of nodes in mesh

con-file String specifying the node connectivity file name

n-elements Number of elements in mesh

| | |
|--------------|---|
| directory | String specifying the directory path |
| element-type | String or symbol specifying Drakon element type |
| drakon-path | String specifying the path to the drakon directory. Default is (<i>logical-path :drakon-binaries</i>) |

| | |
|---|------------|
| convert-aml-loads-ascii-file-to-drakon-binary | [Function] |
|---|------------|

This function converts the AML loads ASCII files to drakon binary format for the Dracon analysis.

Format:

(convert-aml-loads-ascii-file-to-drakon-binary

loads-file directory

&key drakon-path

)

Arguments:

| | |
|------------|---------------------------------------|
| loads-file | String specifying the loads file name |
|------------|---------------------------------------|

| | |
|-----------|--------------------------------------|
| directory | String specifying the directory path |
|-----------|--------------------------------------|

| | |
|-------------|---|
| drakon-path | String specifying the path to the drakon directory. Default is (<i>logical-path :drakon-binaries</i>) |
|-------------|---|

| | |
|---|------------|
| convert-aml-fixed-nodes-ascii-file-to-drakon-binary | [Function] |
|---|------------|

This function converts the AML constraints ASCII files to drakon binary format for the Dracon analysis.

Format:

(convert-aml-fixed-nodes-ascii-file-to-drakon-binary

fixed-nodes-file directory

&key drakon-path

)

Arguments:

fixed-nodes-file String specifying the fixed nodes file name

directory String specifying the directory path

drakon-path String specifying the path to the drakon directory. Default is
(*logical-path :drakon-binaries*)

convert-aml-zero-thickness-elements-ascii-file-to-drakon-binary

[Function]

This function converts the AML zero thickness ASCII files to drakon binary format for the Dracon analysis.

Format:

(convert-aml-zero-thickness-elements-ascii-file-to-drakon-binary

zero-thickness-elements-file directory

&key drakon-path

)

Arguments:

zero-thickness-elements-file String specifying the zero thickness elements file name

directory String specifying the directory path

drakon-path String specifying the path to the drakon directory. Default
is (*logical-path :drakon-binaries*)

run-drakon

[Function]

This function runs for the Dracon analysis.

Format:

(run-drakon

analysis-type directory num-iterations load-cases

&key drakon-path

)

Arguments:

analysis-type Options are :analysis or :optimization to either run the Drakon structural analysis or the Drakon Optimization

directory String specifying the directory path

num-iterations Number of iterations for to run in the optimization

load-cases List of load values

drakon-path String specifying the path to the drakon directory. Default is *(logical-path :drakon-binaries)*

convert-drakon-binary-result-files-to-aml-ascii

[Function]

This function converts the Drakon binary results files to ASCII files for analysis.

Format:

(convert-drakon-binary-result-files-to-aml-ascii

directory aml-displ-prefix aml-stress-prefix

&key drakon-path n-load-cases

)

Arguments:

| | |
|-------------------|---|
| directory | String specifying the directory path |
| aml-displ-prefix | Prefix of the displacement result in string format. |
| aml-stress-prefix | Prefix of the stress result in string format. |
| drakon-path | String specifying the path to the drakon directory. Default is <i>(logical-path :drakon-binaries)</i> |
| n-load-cases | Number of load cases in the file. Defaults to 1. |

drakon-interface-class

[Class]

This class provides the necessary objects and properties to interface AML with the Drakon stress and optimization analysis.

Inherit-from: name-generator

Properties:

| | |
|-------------------------------|--|
| mesh-query | Mesh query instance for the structure being analyzed. Defaults to <i>(default nil)</i> |
| fixed-nodes-query | Mesh query instance of the constrained nodes. Defaults to <i>(default nil)</i> |
| zero-thickness-elements-query | Mesh query instance of the nodes that have zero thickness such as holes or cutouts. Defaults to <i>(default nil)</i> |
| loads-query | Mesh query instance of the loaded nodes. Defaults to <i>(default nil)</i> |
| directory | String specifying the directory path where the analysis files are located. Defaults to <i>(default nil)</i> |
| analysis-type | Options are :analysis or :optimization to either run the Drakon structural analysis or the Drakon |

| | |
|----------------------------------|--|
| | Optimization analysis Defaults to <i>(default nil)</i> |
| n-iterations | Number of iterations for to run in the optimization Defaults to <i>(default 1)</i> |
| load-cases | Number of load cases in the analysis. Defaults to <i>(default 1)</i> |
| element-type | Element type used in the analysis. 3 represents a three sided element, 5 represents a membrane-quad element, and 6 represents a sandwich element. Defaults to <i>(default 5)</i> |
| drakon-binaries-path | String specifying the directory path to the Drakon binaries. Defaults to <i>(logical-path :drakon- binaries)</i> |
| generate-materials-file? | Demanding this property will generate the materials file called mater.dat in the directory specified by the <i>directory</i> property |
| convert-mesh? | Demanding this property will convert the mesh files from AML format to Drakon binary format |
| convert-loads? | Demanding this property will convert the load files from AML format to Drakon binary format |
| convert-fixed-nodes? | Demanding this property will convert the constraint files from AML format to Drakon binary format |
| convert-zero-thickness-elements? | Demanding this property will convert the zero- thickness files from AML format to Drakon binary format |
| run-drakon? | Demanding this property will (if needed) convert the mesh, generate the materials file, convert the loads, convert the fixed nodes, convert the zero thickness elements, run the specified analysis depending on the <i>analysis-type</i> property |
| convert-results? | Demanding this property will run Drakon if necessary and convert the output files to AML format ASCII files |
| displacement-output-file | File containing the displacement results. Defaults |

| | |
|-------------------------|---|
| | to <i>(nth 0 (nth 0 ^convert-results?))</i> |
| stress-output-file | File containing the stress results. Defaults to <i>(nth 0 (nth 1 ^convert-results?))</i> |
| Subobjects: | |
| drakon-results-tables | An instance of mesh-with-color-mapping-tables-class, this object collects the data and results from the Drakon analysis and is used in the displacements and stress visualizations. |
| displacements | A series of mesh-with-color-mapping-class objects showing the displacements from the Drakon analysis |
| equivalent-stress | An instance of mesh-with-color-mapping-class showing the equivalent stress output from the Drakon analysis |
| lcf-equivalent-stress | An instance of mesh-with-color-mapping-class showing the lcf equivalent stress output from the Drakon analysis |
| lcf-equivalent-stress-1 | An instance of mesh-with-color-mapping-class showing the lcf equivalent stress 1 output from the Drakon analysis |
| lcf-equivalent-stress-2 | An instance of mesh-with-color-mapping-class showing the lcf equivalent stress 2 output from the Drakon analysis |
| thickness | An instance of mesh-with-color-mapping-class showing the thickness distribution output from the Drakon analysis |
| stress-vectors | An instance of mesh-with-color-mapping-class showing the vector plot of the stress output from the Drakon analysis |

Drakon Interface Examples

[Class]

These classes are provided as an example to demonstrate the AML/Drakon interface. The

data files can be found electronically in the AML/Drakon interface system.

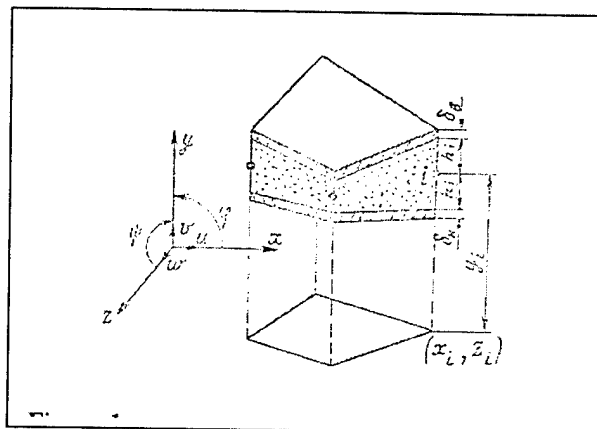
```
(define-class drakon-sandwich-test
  :inherit-from (drakon-interface-class)
  :properties (
    directory          "c:\\temp\\drakon-sandwich\\"
    aml-crd-file       (logical-path ^directory "model.crd")
    aml-con-file       (logical-path ^directory "model.con")
    aml-loads-file     (logical-path ^directory "model.loads")
    aml-fixed-nodes-file (logical-path ^directory "model.displ")
    element-type       (default 6)
  )
)

(define-class drakon-quadmesh-test
  :inherit-from (drakon-interface-class)
  :properties (
    directory          "c:\\temp\\drakon-quad\\"
    aml-crd-file       (logical-path ^directory "model.crd")
    aml-con-file       (logical-path ^directory "model.con")
    aml-loads-file     (logical-path ^directory "model.loads")
    aml-fixed-nodes-file (logical-path ^directory "model.displ")
    element-type       (default 5)
  )
)
```

Appendix 6:
General description of the sandwich plate element used in AML

Alexander Danilin – Samara State Aerospace University
Terrence A. Weisshaar – Purdue University

The sandwich panel element used by AML is shown in Figure 1. This element is a special sandwich plate developed at Samara State Aerospace University, Samara, Russia, especially for structural design optimization; its development will be summarized here so that a user can understand the assumptions used to create the element. This element consists of two isotropic membrane skins with a continuous filler or core between them. This element resists bending, twist and shear loads.



The projection of the element to the x-z plane has the shape of an arbitrary quadrilateral. The lines connecting the top and bottom surfaces of the sandwich are parallel to the y-axis shown in Figure 1.

The filler core is relatively rigid in the transverse (y) direction so that normal strains in the y-direction are very small. The assumption about filler core incompressibility in the transverse direction simplifies the development of the element stiffness matrix and reduces the number of degrees of freedom. The core is shear deformable and distorts as it resists shear stress and transmits these loads between the top and bottom skins. This shear deformation behavior may be isotropic or orthotropic. The stiffness of this filler core is expressed as a matrix

$$G = \begin{bmatrix} G_{yx} & 0 \\ 0 & G_{yz} \end{bmatrix} \quad (1)$$

G_{yz} and G_{yx} are the shear moduli of the filler core in the y-z plane and x-y plane, respectively.

The nodes of the filler element portion are placed at the middle of core element edges on the y-axis, as indicated in Figure 1. Each node has five degrees of freedom: three displacements (u , w , v) and two angles (ϕ and ψ). The element skin thicknesses are δ_b and δ_H , the modulus of elasticity is E and the Poisson's ratio of the skin material is μ . For each node the coordinates x , y , z and element height $2h$ are known.

The skin itself can have curvature. For this reason, each skin panel is divided into four triangles, as shown in Figure 2. In the element derivation the panel stiffness matrix has a contribution from the skin that is accounted for by summing the individual stiffness matrices of the triangles. The coordinates of outside nodes determine the coordinates of the interior node. The selection of this skin model is made for the several reasons.

First of all, the triangle is the simplest membrane element for modeling an isotropic skin, therefore creation of a stiffness matrix and subsequent programming are greatly simplified.

The introduction of an interior node and definition of its position is accomplished using a least squares method; this allows careful approximation of the median surface of the skin using the four triangles.

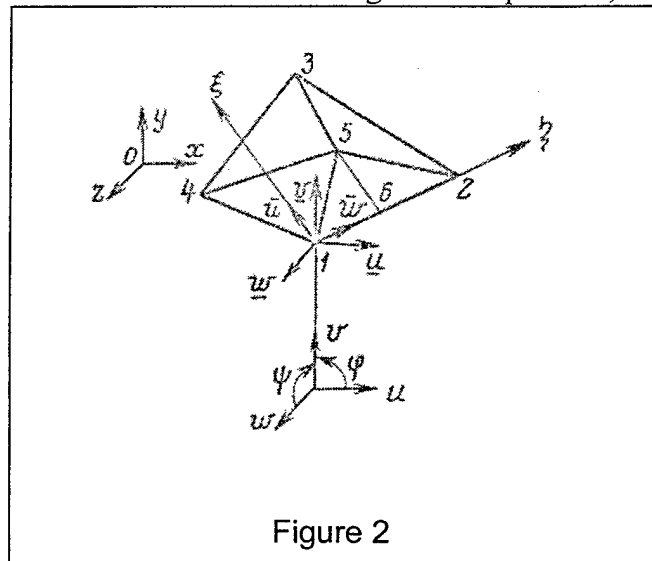


Figure 2

Stiffness matrix of skin panels

To illustrate the features of the element development, let's consider the upper skin panel

shown in Figure 2. Its coordinates on the y-axis are given by

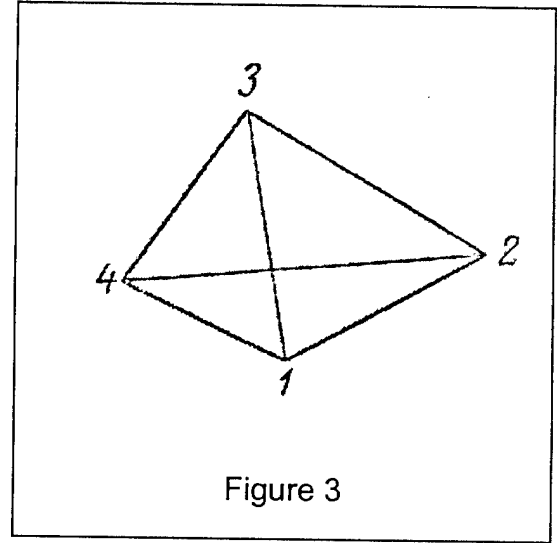
$$H_i = y_i + (h_i + 0.5 \delta_b) \quad i = 1, 2, 3, 4, \quad (2)$$

Coordinates of the interior point 5 are calculated from a least squares method using coordinates of the other four nodes of the upper skin element. To solve for the x-axis coordinate it is necessary

to minimize a function $f(x_5) = \sum_{i=1}^4 (|x_5 - x_i|)^2$.

Equating to zero the derivative of $f(x_5)$ with respect to x_5 and solving this equation, we find

$$x_5 = \frac{1}{4} \sum_{i=1}^4 x_i. \quad (3)$$



The coordinates z_5 and H_5 are similarly calculated.

To understand the development of the upper skin stiffness matrix, consider a triangle 125 in Figure 2. Enter a local coordinate system in the plane of a triangle with the beginning at the point 1. An axis h is placed through link 1-2, and an axis x is placed perpendicular to h . The length of segment 1-2 is then

$$d_{12} = \sqrt{x_{21}^2 + z_{21}^2 + H_{21}^2}, \quad (4)$$

Where $x_{21} = x_2 - x_1$; $z_{21} = z_2 - z_1$; etc.

Direction cosines of the axis h with respect to axis x,y,z are as follows

$$l_{12} = \frac{x_{21}}{d_{12}}; \quad m_{12} = \frac{H_{21}}{d_{12}}; \quad n_{12} = \frac{z_{21}}{d_{12}}. \quad (5)$$

For the definition of direction cosines of an axis x from point 5 we drop a perpendicular to the axis h. The length of segment 1-6 is

$$d_{16} = l_{12} x_{51} + m_{12} H_{51} + n_{12} z_{51}, \quad (6)$$

and the length of a segment 6-5 is

$$d_{65} = \sqrt{x_{51}^2 + H_{51}^2 + z_{51}^2 - d_{16}^2}. \quad (7)$$

Direction cosines of the axis x are

$$l_{65} = \frac{x_{51} - l_{12}d_{16}}{d_{65}}; \quad m_{65} = \frac{H_{51} - m_{12}d_{16}}{d_{65}}; \quad n_{65} = \frac{z_{51} - n_{12}d_{16}}{d_{65}}. \quad (8)$$

Local nodal coordinates of the triangle are written as

$$\bar{X} = \bar{\lambda}(X - X_I) \quad (9)$$

where

$$\bar{X} = \begin{bmatrix} \xi_1 & \xi_2 & \xi_3 \\ \eta_1 & \eta_2 & \eta_3 \end{bmatrix}; \quad \bar{\lambda} = \begin{bmatrix} l_{65} & m_{65} & n_{65} \\ l_{12} & m_{12} & n_{12} \end{bmatrix}$$

$$X = \begin{bmatrix} x_1 & x_2 & x_3 \\ H_1 & H_2 & H_3 \\ z_1 & z_2 & z_3 \end{bmatrix}; \quad X_1 = \begin{bmatrix} x_1 & x_1 & x_1 \\ H_1 & H_1 & H_1 \\ z_1 & z_1 & z_1 \end{bmatrix}.$$

The displacements inside the triangle are written as

$$\bar{u} = a_0 + a_1\xi + a_2\eta, \quad \bar{w} = b_0 + b_1\xi + b_2\eta \quad (10)$$

In a matrix form these are written as

$$\begin{Bmatrix} \bar{u} \\ \bar{w} \end{Bmatrix} = [d] \cdot \{a\} \quad (11)$$

where

$$\{a\} = \begin{Bmatrix} a_0 \\ a_1 \\ a_2 \\ b_0 \\ b_1 \\ b_2 \end{Bmatrix}; \quad [d] = \begin{bmatrix} 1 & \xi & \eta & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & \xi & \eta \end{bmatrix}$$

Using boundary conditions, we compute the displacements of the vertices of the triangle to be

$$\bar{U} = D \cdot \{a\} \quad (12)$$

where

$$\bar{U} = \begin{Bmatrix} \bar{u}_1 \\ \bar{w}_1 \\ \bar{u}_2 \\ \bar{w}_2 \\ \bar{u}_5 \\ \bar{w}_5 \end{Bmatrix}; \quad D = \begin{bmatrix} 1 & \xi_1 & \eta_1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & \xi_1 & \eta_1 \\ 1 & \xi_2 & \eta_2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & \xi_2 & \eta_2 \\ 1 & \xi_5 & \eta_5 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & \xi_5 & \eta_5 \end{bmatrix}.$$

Solving the matrix equation (12) and substituting this expression into Eqn. (11), we find

$$\begin{Bmatrix} \bar{u} \\ \bar{w} \end{Bmatrix} = [d][D]^{-1}\bar{U}. \quad (13)$$

Strains in the triangular panel are then found to be

$$\varepsilon = \begin{Bmatrix} \varepsilon_\xi \\ \varepsilon_\eta \\ \gamma_{\xi\eta} \end{Bmatrix} = \begin{Bmatrix} \frac{\partial \bar{u}}{\partial \xi} \\ \frac{\partial \bar{w}}{\partial \eta} \\ \frac{\partial \bar{w}}{\partial \xi} + \frac{\partial \bar{u}}{\partial \eta} \end{Bmatrix} \quad (14)$$

or

$$\varepsilon = b \cdot \bar{U} \quad (15)$$

where

$$b = \frac{1}{2A_{125}} \begin{bmatrix} \eta_{52} & 0 & -\eta_{51} & 0 & \eta_{21} & 0 \\ 0 & -\xi_{52} & 0 & \xi_{51} & 0 & -\xi_{21} \\ -\xi_{52} & \eta_{52} & \xi_{51} & -\eta_{51} & -\xi_{51} & \eta_{21} \end{bmatrix} \quad (16)$$

and $2A_{125} = \xi_{52}\eta_{21} - \xi_{21}\eta_{52} \quad (17)$

Translational displacements in the plane of a triangle are calculated as

$$\bar{U} = \lambda \underline{U} \quad (18)$$

where

$$\underline{U} = \{ \underline{u}_1 \quad \underline{w}_1 \quad \underline{v}_1 \quad \underline{u}_2 \quad \underline{w}_2 \quad \underline{v}_2 \quad \underline{u}_5 \quad \underline{w}_5 \quad \underline{v}_5 \}^T, \quad \lambda = \begin{bmatrix} \bar{\lambda} & 0 & 0 \\ 0 & \bar{\lambda} & 0 \\ 0 & 0 & \bar{\lambda} \end{bmatrix}$$

Translations $\{ \underline{u} \quad \underline{w} \quad \underline{v} \}^T$ are computed using translations $\{ u \quad w \quad \varphi \quad \psi \quad v \}^T$

$$\underline{u} = u - \varphi h'; \quad \underline{w} = w - \varphi h'; \quad \underline{v} = v \quad (19)$$

where $h' = h + 0.5\delta_b \quad (20)$

Nodal displacements are

$$\underline{U} = \Pi \tilde{U} \quad (21)$$

where $\tilde{U} = \{u_1 \ w_1 \ \varphi_1 \ \psi_1 \ v_1 \ u_2 \ w_2 \ \varphi_2 \ \psi_2 \ v_2 \ u_5 \ w_5 \ \varphi_5 \ \psi_5 \ v_5\}^T$;

$$\Pi = \begin{bmatrix} \Pi_1 & 0 & 0 \\ 0 & \Pi_2 & 0 \\ 0 & 0 & \Pi_5 \end{bmatrix}; \quad \Pi_i = \begin{bmatrix} 1 & 0 & -h'_i & 0 & 0 \\ 0 & 1 & 0 & -h'_i & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}, \quad i = 1, 2, 5.$$

The displacements of the interior node are calculated as the mean arithmetical displacements of the four nodes

$$\tilde{U} = B \cdot U \quad (22)$$

where

$$U = \{u_1 \ w_1 \ \varphi_1 \ \psi_1 \ v_1 \ u_2 \ w_2 \ \varphi_2 \ \psi_2 \ v_2 \ u_3 \ w_3 \ \varphi_3 \ \psi_3 \ v_3 \ u_4 \ w_4 \ \varphi_4 \ \psi_4 \ v_4\}$$

Doing this for other triangles produces a banded B matrix like that shown in Figure 4.

Substituting Eqn. (13) into Eqn. (15) and solving Eqns. (21) and (22), we calculate strains and stress in the triangle 125 as

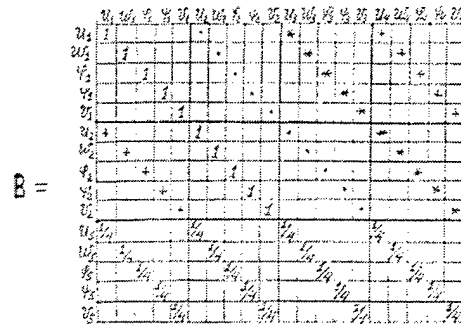


Figure 4 – Banded B matrix

$$\varepsilon = L \cdot U, \quad (23)$$

$$\sigma = \{\sigma_{\xi} \sigma_{\eta} \sigma_{\xi\eta}\} = \kappa \cdot L \cdot U, \quad (24)$$

$$\text{where } L = b\lambda\Pi B, \quad \kappa = \frac{E}{1-\mu^2} \begin{bmatrix} 1 & \mu & 0 \\ \mu & 1 & 0 \\ 0 & 0 & \frac{1-\mu}{2} \end{bmatrix}. \quad (25)$$

The stiffness matrix contribution for an element is obtained by integration of the expression $L^T \kappa L$ over the element volume. For the triangular panel 125 the stiffness matrix is written as

$$K = L^T \kappa L |A_{125}| \delta_b \quad (26)$$

The stiffness matrix of the upper skin panel is obtained by summation. The stiffness matrices of the four triangles are computed, using equations similar to Eqns. (4) - (26), and then summed.

The stiffness matrix of the lower panel is computed by the same formulas. To do this it is necessary to change the sign before brackets in Eqn. (2) and before h_i in Eqn. (19) and to substitute δ_H instead of δ_b in Eqn. (2) and Eqn. (20).

Skin panel stresses

We calculate panel stresses by first substituting the coordinates of panel nodes

$$\bar{Y} = C^T (CC^T)^{-1} CY \quad (27)$$

$$\bar{h} = C^T (CC^T)^{-1} Ch \quad (28)$$

where $\bar{Y} = \{\bar{y}_1 \bar{y}_2 \bar{y}_3 \bar{y}_4\}$ are the coordinates of the plane panel and $Y = \{y_1 y_2 y_3 y_4\}$ are
 $\bar{h} = \{\bar{h}_1 \bar{h}_2 \bar{h}_3 \bar{h}_4\}$ $h = \{h_1 h_2 h_3 h_4\}$

coordinates of the curved panel, and $C = \begin{bmatrix} 1 & 1 & 1 & 1 \\ x_1 & x_2 & x_3 & x_4 \\ z_1 & z_2 & z_3 & z_4 \end{bmatrix}$.

The stresses in each triangle are defined by Eqn. (24). Thus in the formulas for an evaluation of the matrices B also it is necessary to substitute coordinates obtained from Eqns. (27) and (23). The stresses in the panel are easily calculated by simply averaging of stresses in the triangles.

Stiffness matrix of filler core

The linear relationship between filler core vertical displacements and nodal displacements is

$$v = c_0 + c_1 x + c_2 z \quad (29)$$

Shear strains occur because of rotation of rigid vertical lines in the core. We shall calculate as mean arithmetical rotation angles the following

$$\gamma = \{\gamma_{yx} \gamma_{yz}\} = \left\{ -\frac{1}{4} \sum_{i=1}^4 \varphi_i + \frac{\partial v}{\partial x}; -\frac{1}{4} \sum_{i=1}^4 \psi_i + \frac{\partial v}{\partial z} \right\} \quad (30)$$

$$\text{or } \gamma = \left\{ c_1 - \frac{1}{4} \sum_{i=1}^4 \varphi_i; c_2 - \frac{1}{4} \sum_{i=1}^4 \psi_i \right\} \quad (31)$$

Vertical displacement of element nodes are

$$\bar{V} = \{v_1 v_2 v_3 v_4\} = C^T n \quad (32)$$

where $n = \{c_0 c_1 c_2\}$.

From Eqn. (32), using a least squares method, we find

$$n = R \bar{V} \quad (33)$$

$$\text{where } R = (CC^T)^{-1} C. \quad (34)$$

Then strains and stresses in the core are

$$\gamma = L_f U \quad (35)$$

$$\tau = \{\tau_{yx} \tau_{yz}\} = \bar{G} \gamma \quad (36)$$

where $L_f = mRA + T$ (37)

and $m = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$

The stiffness matrix of the core is

$$k_f = L_f^T \bar{G} L_f V \quad (38)$$

where the volume of the core is

$$V = \frac{1}{4} A_{1234} \sum_{i=1}^4 2h_i \quad (39)$$

and $A_{1234} = \frac{1}{2} (|x_{32}z_{21} - x_{21}z_{32}| + |x_{34}z_{31} - x_{31}z_{34}|) \quad (40)$

is the square of a projection of the element on the plane. The stiffness matrix of all elements is found by summing the stiffness matrices from both the top and bottom skin panels and the stiffness matrix of the core filler.

Appendix 7:
AML Dracon Interface Documentation The Use of Optimality Criteria for Aircraft
Conceptual Level Structural Design

Alexander I. Danilin*
Terrence A. Weisshaar†

Abstract

This paper illustrates the use of established optimality criteria with deflection constraints to add information during the early design stage to aid in a multi-disciplinary, as opposed to a structural design, effort. Our objectives are three: to estimate weight of unusual design components accurately; to provide stiffness information when aeroelastic or other stiffness constraints are involved; and, to provide a seamless path from conceptual to preliminary design where accurate, gradient-based optimization methods can be applied. This approach requires a simple, but accurate, method to address static aeroelastic constraints such as lifting surface effectiveness. We address aeroelastic constraints by furnishing a rationale for locating typical sections on lifting surfaces (such as was first done in an *ad hoc* manner by Theodorsen). Our examples indicate that control of sectional deformation of parts of lifting surfaces lying between 70% and 90% of the semi-span will serve as a surrogate for precise aeroelastic control. We also discuss how to identify structural locations where we are likely to find minimum gage thicknesses. Our approach is illustrated with two problems: design of a vertical tail with and without a lift effectiveness constraint; and, design of a supersonic transport wing to reduce trim drag.

Background

The purposes of this paper include: a historical perspective on structural design efforts that

have occurred in Russia during the past decade; and, suggesting how to improve structural conceptual design by reducing risk, adding fidelity and encouraging creativity. This improvement includes the routine use of formal optimization methods for structural conceptual design at the earliest possible time. Our process depends on using optimality criteria for structural design; this area is a reasonably well-developed analytical technology. Schmidt[1] gives a comprehensive list of early papers in optimality criteria that is sufficient for our purposes.

Most optimality criteria focus on fully stressed design. All optimality algorithms for solution use recursive redesign rules. One limitation of fully stressed design is that it assumes that a fully stressed design is equivalent to a minimum weight design. This is not a universal rule and counter examples abound. We do not wish to revisit these arguments, except to say that the particular redesign algorithm used here is based on designing short load paths into the structure[2]. The

* Professor, Samara State Aerospace University, Samara, Russia

† Professor, School of Aeronautics and Astronautics, Purdue University, West Lafayette, Indiana, Fellow AIAA.

Copyright © 2000 by Alexander I. Danilin and Terrence A. Weisshaar. Published by the American Institute of Aeronautics and Astronautics, Inc. with permission

particular strength of our method is that it uses a finite element mesh that, if fine enough, can be used later for preliminary design, in which general mathematical programming approaches can be exercised on appropriate models. When this is done, we have a relatively seamless boundary between primitive conceptual and preliminary design models.

In a design activity there are always two tasks, analysis and synthesis. Our effort is intended to bring these two areas closer together. We have found that if we do not contrast the difference between analysis and synthesis at the outset, then some of our points will be lost. In our context, analysis is "the activity directed toward the installation of links between parameters and characteristics of an object that has already been designed." The definition of the maximum stress in a wing after all of its geometrical parameters have been selected is one such example. We note that such a task can be solved both by a mathematical simulation and by a full-scale experiment, so our definition of analysis also extends to testing activities.

On the other hand, synthesis is "the activity directed toward the choice of a structural form and such parameters for the designed object that minimize (or maximize) one or more of its characteristics and satisfy all restrictions." For example, the choice of the number and arrangement of spars and ribs and the materials to use to make these elements is a product of structural synthesis. In the majority of cases the

task of synthesis becomes the prerogative of the creative abilities of human intellect and experience. This does not mean that humans always make good decisions or that they cannot be helped by analysis.

In the United States there are several approaches to weight minimization with aeroelastic constraints. Certainly the most recognized and venerable of these codes are TSO [cf. Ref. 3] and FASTOP [cf. Ref. 4], codes that date from the 1970's. FASTOP most resembles the approach used here although it is more accurate, but requires more design definition to operate and thus allows less latitude for design efforts. More recently, ASTROS[5] has become a standard for advanced design optimization with aeroelastic constraints.

Still, at the beginning of the design process, particularly those that focus on new concepts, focus is seldom sharp. Our concept addresses this lack of clarity and premium on understanding of new concepts. Figure 1 shows an example of how analysis and

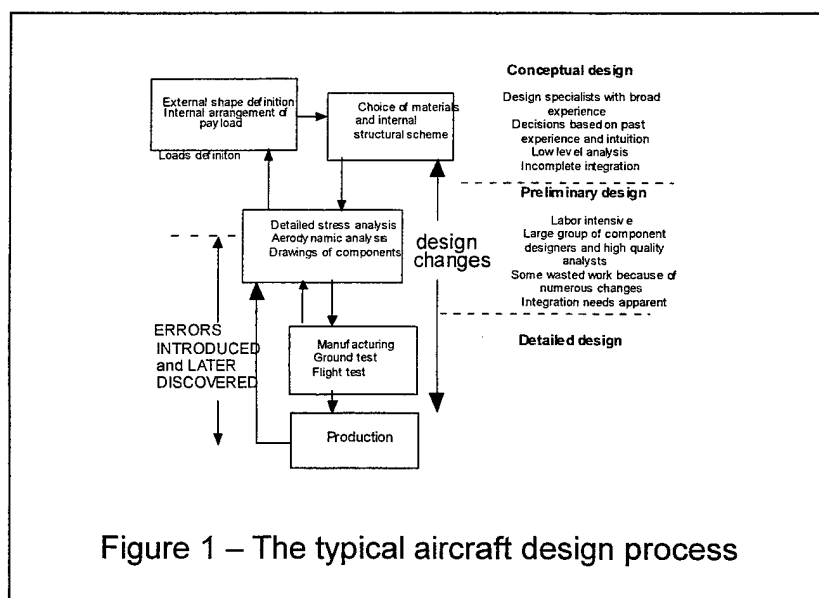


Figure 1 – The typical aircraft design process

synthesis are organized for a design activity. There is a need to reorganize this process to take advantage of advances in computation and organizational dynamics[6]. For one thing, the level of structural analysis used in conceptual design is usually low.

If the organization always builds the same time of airplane for the same purpose, our approach will not be very useful. If the choices are few the team needs analysis, not synthesis. We believe that the secret of successful creative, concurrent design is the blending of analysts and designers, but we defer this extended discussion to others[7].

Many articles have been written about the design process and the move towards concurrent design[8]. We will not review these articles here, except to say that it is a challenge to include the structural design to the same extent as aerodynamic design that can define outer mold-lines with exquisite accuracy. This aerodynamic synthesis/analysis has been fueled by analytical capabilities in aerodynamic paneling codes and CFD. Structural analysis can also support design, but it is bereft of definitions required to generate models until it is sometimes too late to make creative contributions.

Structural design is loads driven. The thousands of loads that define the myriad of skeletal dimensions such as panel thicknesses cannot be defined until well into the design process. As a result, the chance of using creative concepts diminishes as other members of the design team add features. This is particularly true in the case of concepts

that rely on exploitation of aeroelastic design features. These aeroelastic tailoring concepts are stiffness dependent and the decision, early in the design process; to place a main spar at a specific point, in a certain direction may also ensure that a concept, such as an active aeroelastic wing, will not have a strong benefit.

During conceptual, structures related design, discussions about weight are serious. For instance, the advocacy of a new concept on the basis of reduced cost or weight is often on "thin ice" and sometimes relies more on hope than hard numbers or on the "experience" or "feelings" of the advocate; these feelings are often not shared by others. We believe that our approach helps to provide hard evidence that can be generated before, not after, the structural layout is locked into the process. Among our limited objectives is the use of simple optimality criteria and "typical" sections to resolve aeroelastic issues as early as possible.

Identification of aeroelastic characteristic (typical) sections

Some aeroelastic stiffness requirements such as lift effectiveness can be posed approximately as a limitation on deformation (note that this "limitation" can also be a "requirement" for flexibility). To use optimality criteria, these constraints enter as inequalities and, in the case of wings, the measure of design stiffness is taken as an elastic displacement at a finite number of selected, "characteristic" or "typical" cross-sections.

The deformations of these wing cross-sections and their aerodynamic properties are representative of the elastic twist angle $\theta(\bar{z})$ and the chordwise change of curvature (camber) of the structure $\Delta\bar{f}(\bar{z})$ along the wing; the coordinate \bar{z} is the nondimensional distance from the lifting surface root. From design experience it is known that θ and $\Delta\bar{f}$ contribute approximately the same effects when

$$2 \Delta\bar{f} \cong \theta \quad (1)$$

For large-aspect-ratio wings with sections having thickness-to-chord ratios between 10% and 12%, the change in camber curvature is insignificant compared to the elastic twist, and the condition given in Eqn. 1 is not found at any spanwise cross-section.

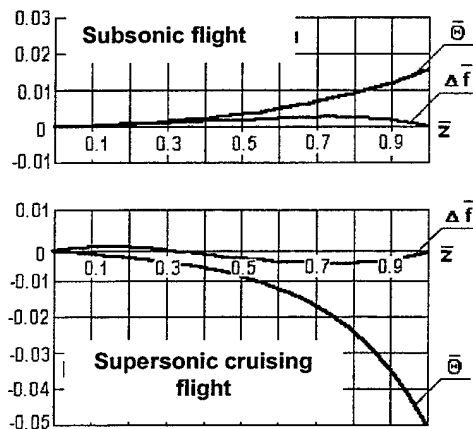


Figure 2 - Chordwise curvature and twist for an example low aspect ratio transport wing; subsonic and supersonic flight

For low-aspect-ratio, high-speed wings with thickness-to-chord ratios between 3% and 5%, relatively large chordwise deformation occurs

so that is possible to have a ratio like that shown in Eqn. 1 on some sections. In Figure 2 the curves $\theta(\bar{z})$ and $\Delta\bar{f}(\bar{z})$ for a supersonic passenger airplane wing with an ogival planform with thickness ratio 5% are shown. It is seen that elastic twist dominates the airloads at $\bar{z} > 0.5$.

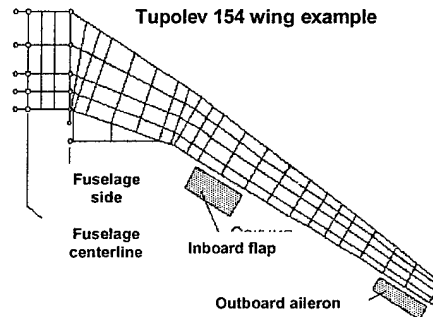


Figure 3 - Tu-154 structural finite element model

Theodorsen described these circumstances when he considered torsional deformations of a characteristic section at the $3/4$ spanwise position to be "characteristic" of the behavior of the wing in general. He did not, to our knowledge, publish the guidelines used to select his typical section. Other researchers recommend a characteristic cross-section located at different \bar{z} locations. Most recommend selecting a characteristic cross-section in the range $\bar{z} = 0.7$ to 0.9 .

References, which substantiate the principles used to select a characteristic cross-section, are not known to us. Therefore we conducted a study to identify how to select such a section.

First we consider the high-aspect wing shown in Figure 3. This wing has an aerodynamic aspect ratio $AR = 7.5$; only the load-carrying part of a design is shown. The structural distribution and the geometrical

characteristics of model correspond to the wing of the Tupelov 154 aircraft. Let's determine the torsional deformations of cross-sections for three loading conditions:

1. loading by tip section ailerons
2. loading by an inboard flap
3. a case with a forward center of pressure

Because the relative twist, not the absolute value of the twist, is important, we will examine two different normalized relationships:

$$\bar{\theta}_1(\bar{z}) = \frac{\theta(\bar{z})}{\theta(1)} \quad (2)$$

$$\bar{\theta}_2(\bar{z}) = \frac{\theta(\bar{z})}{\int_0^1 \theta(\bar{z}) d\bar{z}} \quad (3)$$

Figure 4 plots these two relationships. It is difficult to draw practical conclusions from a plot of $\bar{\theta}_1(\bar{z})$, but the curves $\bar{\theta}_2(\bar{z})$ show an interesting feature. These three curves intersect very close to each other at $\bar{z} = 0.75$. We notice that for all three loading conditions we have

$$\bar{\theta}_2(0.75) = \frac{\theta(0.75)}{\int_0^1 \theta(\bar{z}) d\bar{z}} \cong \text{const.} \quad (4)$$

From this result it follows that the twist at $\bar{z} = 0.75$ is

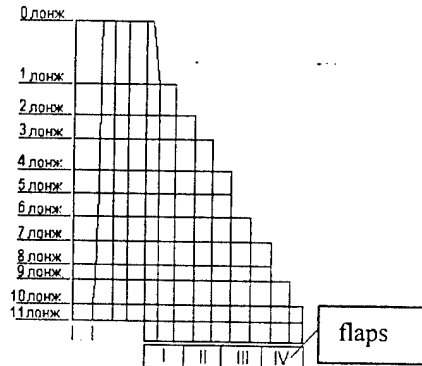


Figure 5 – Structural finite element model – note coarse mesh

$$\theta(0.75) \cong \text{const} \int_0^1 \theta(\bar{z}) d\bar{z}. \quad (5)$$

This cross-section location is in the range \bar{z} first mentioned by Theodorsen as his selection

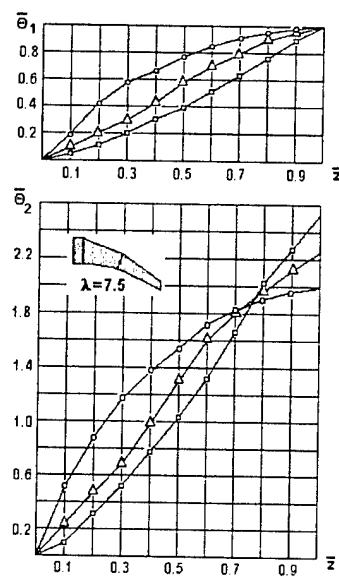


Figure 4. Twist deflection results for:
o – flap loading;
 Δ – forward center of pressure;
 \square – aileron loading.

for the characteristic section.

Let's try this approach for the low aspect ratio transport wing whose coarse finite element structural model is shown in Figure 5. The five loads applied to the wing are: a subsonic cruise condition; take-off; aileron loading on element IV; a twisting moment; and, aileron loading on section II. The results shown in Figure 6 demonstrate that here too it is possible to find a characteristic section, in this case at $\bar{z} = 0.83$.

As a result of this study, propose the following simple technique to find the typical section for optimization studies. The twist distribution on a "typical" wing is computed for several loading conditions. The curves $\bar{\theta}_2(\bar{z})$, defined in Eqn. 3, are constructed and the approximate point of their intersection determines the position of the characteristic section. This gives us an empirical guide for optimization studies.

Stiffness constraints and optimality criteria

To enforce an approximate aeroelastic constraint we need a method for constraining displacements, particularly those related to surface rotation. Consider the search for a structural design with minimal volume (mass) with a generalized displacement constraint at a single point on the structure. To develop our constraint relationship, we first apply a unit generalized force (unit load) in the direction of the constrained displacement (either a displacement or a rotation such as twist). We use the Maxwell-Mohr formulas to calculate the constrained displacement, called Δ_o . This is also referred to as the unit load or dummy load method formula. We first assume that the design is divided into m finite elements, each in a plane stress condition, so that

$$\Delta = \sum_{i=1}^m \int_{S_i} \frac{[R_i^*]}{E_i \delta_i} dS_i, \quad (6)$$

$$[R_i^*] = [\bar{R}_{xi}(R_{xi} - \mu_i R_{yi}) + \bar{R}_{yi}(R_{yi} - \mu_i R_{xi}) + 2(1 + \mu_i)\bar{R}_{xyi}R_{xyi}]$$

Here $\bar{R}_i = \bar{\sigma}_i \delta_i$ represent internal forces in element i due to the unit loading; $R_i = \sigma_i \delta_i$ represents internal forces in element i due to the applied loading; δ_i is the thickness of the element; S_i is the planar area of the element; E_i and μ are the modulus of elasticity and Poisson's ratio of the element material.

The internal loads due to the unit generalized force and applied loading are called \bar{R}_i and R_i ; they are determined by the usual finite element procedure. If finite elements with a constant stress field are used, Eqn. 6 becomes:

$$\Delta = \sum_{i=1}^m \frac{[R_i^*] S_i}{E_i \delta_i} \quad (7)$$

Equation 7 can be interpreted for rod elements; S_i is the length of the rod; δ_i is the cross-sectional area; and, $[R_i^*]$ is the product of internal forces of the basic and unit loading cases.

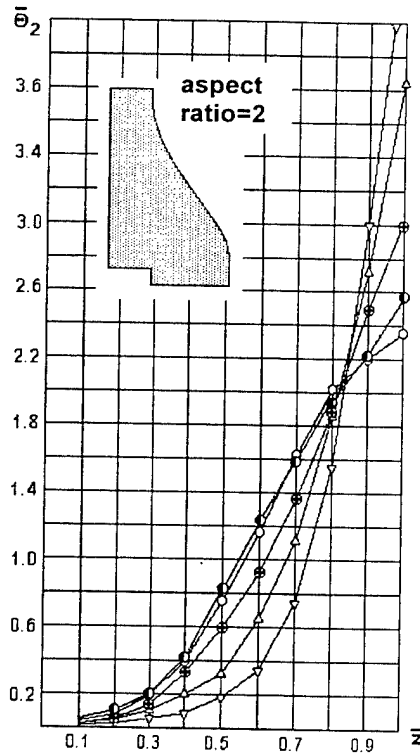


Figure 6 – Torsional deformation distribution for low aspect ratio supersonic transport wing

For thin-walled structures whose elements are in a plane stress condition, the volume of material is written as

$$V = \sum_{i=1}^m S_i \delta_i. \quad (8)$$

We minimize this volume under the condition that

$$\Delta = \Delta_o \quad (9)$$

and
$$V = \sum_{i=1}^m S_i \delta_i \Rightarrow \min \quad (10)$$

where Δ_o is the given value of the generalized displacement.

The elements of the series in Eqn. 7 provide the contribution of each element to the displacement, which is constrained. If the value $[R_i^*]$ is large, the displacement is largely determined by deformations of the i th element. If the size $[R_i^*]$ is negative then reducing the material size or modulus will reduce the deflection. Because the unit generalized force is in the direction of the constrained deformations, a negative value of the Mohr integral shows where it is necessary to reduce the volume of an element.

For an illustration of this method, consider a cantilever beam loaded with two forces, as shown in Figure 7. The tip bending displacement angle θ in Figure 6 is to be constrained to be zero. First apply a unit bending moment $\bar{M}=1$ to the end of the beam in the $+\theta$ direction and find the Mohr integral, defined in this case as

$$I_M = \int_L \frac{\bar{M}M}{EI} dz \quad (11)$$

Assume for simplicity that $EI=\text{constant}$. By multiplying the distribution of the moments from the applied and unit load cases we find the distribution of Mohr integrals over the length of the beam shown in Figure 8.

From the distributions in Figure 8 it is seen that the reduction in bending stiffness between points 2 and 3 will cause an increase in the angle θ , while the reduction of stiffness in site 1 will result in a reduction of θ . The angle θ of the tip section can be reduced by stiffening or strengthening zone 2-3, or destiffening zone 1, or taking these actions simultaneously.

Negative values of the Mohr integrals always identify design zones that are can be "weakened" to satisfy displacement constraints. From Eqn. 7 we see that in these regions it is advisable to choose the minimum thickness allowed for strength, construction or other technological reasons. In Eqn. 7 we collect all terms bearing a negative term so that the equation is written as

$$\Delta = \sum_{i=1}^n \frac{[R_i^*] S_i}{E_i \delta_i} - \sum_{i=n+1}^m \left| \frac{[R_i^*] S_i}{E_i \delta_i} \right| \quad (12)$$

Here δ_i is the minimum allowable thickness of elements with negative contributions to the Mohr integrals; $\tilde{\delta}_i$ is the thickness of elements for which the Mohr integrals are positive; n is the number of elements where the Mohr integrals are positive.

1. We define two terms

$$\Delta^+ = \sum_{i=1}^n \frac{[R_i^*] S_i}{E_i \tilde{\delta}_i}, \quad (13)$$

$$\Delta^- = \sum_{i=n+1}^m \left| \frac{[R_i^*] S_i}{E_i \delta_i} \right| \quad (14)$$

Then

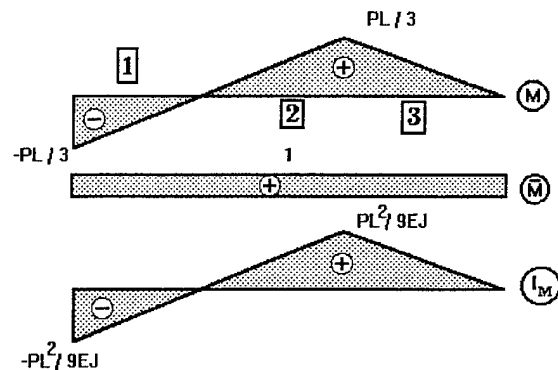


Figure 8 – Internal loading and Mohr integral values

$$\Delta = \Delta^+ - \Delta^- \quad (15)$$

The condition in Eqn. 9 will look like

$$\Delta^+ = \Delta_o + \Delta^- \quad (16)$$

If, in zones with negative Mohr integrals, the minimally allowable thicknesses are used, the thickness of these elements are eliminated as design variable and we find optimum distribution of a material only in zones with positive Mohr integrals. Thus, we have a task of conditional optimization: to minimize the volume of material

$$\tilde{V} = \sum_{i=1}^n S_i \tilde{\delta}_i \Rightarrow \min \quad (17)$$

under the condition in Eqn. 16.

2. LaGrange multipliers are used to find the solution.
Let's define a function

$$L = \sum_{i=1}^n S_i \tilde{\delta}_i + \lambda_1 (\Delta^+ - \Delta_o - \Delta^-) \quad (18)$$

$$\tilde{\delta}_i = \frac{\left\{ \sum_{i=1}^n S_i \sqrt{\frac{[R_i^*]}{E_i}} \right\}}{(\Delta_o + \Delta^-)} \cdot \sqrt{\frac{[R_i^*]}{E_i}} \quad i = 1, 2, \dots, n. \quad (23)$$

where λ_1 is the LaGrange multiplier. The conditions of a minimum of the function are:

$$\frac{\partial L}{\partial \tilde{\delta}_i} = 1 - \lambda_1 \frac{[R_i^*]}{E_i \tilde{\delta}_i^2} = 0, \quad i = 1, 2, \dots, n \quad (19)$$

$$\frac{\partial L}{\partial \lambda_1} = \Delta^+ - \Delta_o - \Delta^- = 0 \quad (20)$$

From Eqns. 19 and 20 we have

$$\tilde{\delta}_i = \sqrt{\lambda_1 \frac{[R_i^*]}{E_i}} \quad (21)$$

Substituting Eqn. 21 into Eqn. 20 we find the LaGrange multiplier

$$\lambda_1 = \frac{\left\{ \sum_{i=1}^n S_i \sqrt{\frac{[R_i^*]}{E_i}} \right\}^2}{(\Delta_o + \Delta^-)^2} \quad (22)$$

Equation 21, accounting for Eqn. 22, can be written as

The expression in Eqn. 23 defines the “law of distribution” of a material for elements of the design ensuring the constraint on the generalized displacement with internal forces \bar{R}_i and R_i .

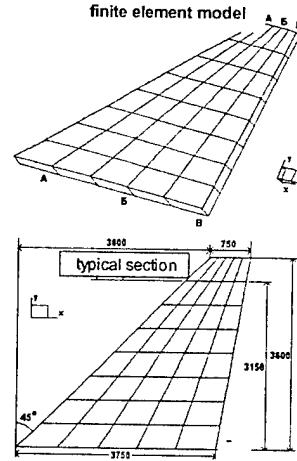


Fig. 10 – Vertical tail finite element model. Spars are indicated as A, b, B. This model has 108 nodes, 157 elements and 321 degrees of freedom.

Let's calculate the required volume of the design from Eqn. 8 by substitution of $\tilde{\delta}_i$ from Eqn. 23 and the minimally allowable thickness δ_i .

$$V = \frac{\left\{ \sum_{i=1}^n S_i \sqrt{\frac{[R_i^*]}{E_i}} \right\}^2}{(\Delta_o + \Delta^-)} + \sum_{i=n+1}^m S_i \delta_i. \quad (24)$$

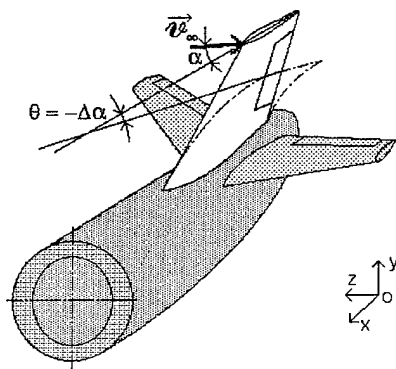


Figure 9 – Vertical tail example

Let's examine Eqns. 23 and 24. In the presence of zones with negative Mohr integrals, that is at $\Delta^- \neq 0$, we have an opportunity to develop a design with a constrained zero generalized displacement: $\Delta_o = 0$. Moreover, the satisfaction of the requirement $\Delta_o < 0$ is possible, provided that $|\Delta_o| < \Delta^-$. If the sites with negative Mohr integrals are absent, we cannot reduce the existing generalized displacement.

Example - vertical tail design with strength and Stiffness constraints

Consider the task of designing of a vertical tail structure with yaw loading, but with the constraint that it be lift effective. The assumed distribution of this aerodynamic loading is assumed to be uniform with an intensity that depends on flight speed and yaw angle. The tail is sweptback so that bending deformation causes negative streamwise angle of attack and reduces the effectiveness of the vertical tail to produce a restoring moment when the tail is displaced sideways (see Figure 9). Our design problem includes a requirement for strength and a constraint on stiffness so

that the tail remains effective. This latter constraint is an aeroelastic constraint. The issue here is how we can increase the effectiveness of the tail by structural redesign and what it will cost or save in terms of weight.

Our aeroelastic constraint, rather than directly addressing aeroelastic effects all along the entire tail surface, will instead require that there be a non-negative angle of twist of a characteristic section located at distance 87.5% of the distance from the root to the tail tip. Thus, the stiffness constraint requires that twist of vertical tail under its aerodynamic loading should be

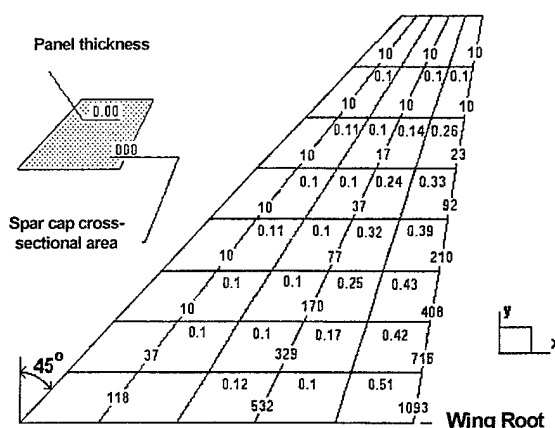


Figure 12 - Final parameter values found from optimization with strength and stiffness constraints; parameter values with minimum gage are not shown on the figure.

opposite to its "natural" deformation.

Our example has a 3-spar vertical tail structure whose coarse geometrical finite element grid is shown in a Fig. 10. Ribs are located along the streamwise direction with uniform spacing; the thickness to chord ratio of the structure is 16 %. Using a software package DRACO (developed by the first author and used extensively for structural design and design education in Russia) the vertical tail structure was optimized.

The initial structural arrangement has uniform distributions of material in elements within each of five structural element groups. The initial skin element thickness is 1 mm; spar web thicknesses are 2 mm; rib webs and leading edge webs are 1 mm; cross sectional areas of rib caps are 100 mm²; spar caps are 1000 mm². The lower limits for the element cross-section areas and

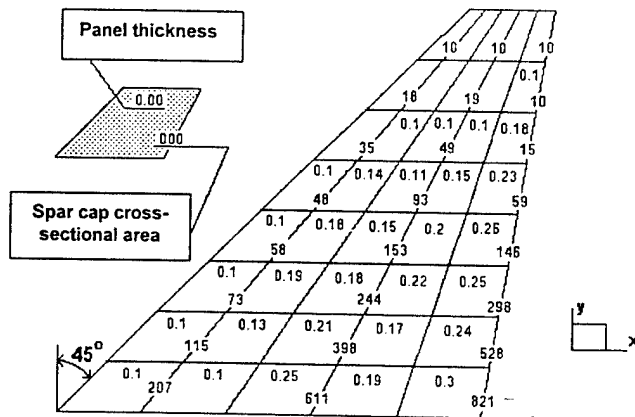


Figure 11 - Final values found from optimization with a strength constraint only

thicknesses are equal to 0.1 % of the initial values. The allowable stress is identical for all elements and equal to 100 N/mm^2 ; the modulus of elasticity of the structural material is $7.2 \times 10^4 \text{ N/mm}^2$. The generalized displacement constraint is $\Delta_o = 0$. The streamwise twist at the position 87.5% from the root of the vertical tail.

If we take into account only strength and minimum gage constraints then, after three design iterations, the distribution of a material shown in Figure 11 is found. The characteristic section has a streamwise twist angle, $\Delta_o = -41.15 \times 10^{-4}$ radians; the volume of structural material is $V = 0.868421 \times 10^6 \text{ mm}^3$.

When we include both the strength condition and the stiffness requirement (streamwise rotation is near zero), after four iterations the distribution of material shown in Figure 11 is found. The streamwise twist angle of the characteristic section is $\Delta_o = +4.44 \times 10^{-4}$ radians. This angle is not quite zero, but it is slightly positive and increases, rather than decreases, effectiveness. The distribution satisfies all imposed strength constraints and

has volume $V_S = 0.876285 \times 10^6 \text{ mm}^3$. This is only 0.9 % heavier than the design with only a strength constraint.

This type of problem has also been considered, independently, by Sensburg and Tischler, in a study in which they used ASTROS and laminate design to improve the effectiveness of a vertical fin[9]. Their results, based on more accurate flight loads also indicate that lift effectiveness can be greatly improved by redistribution and reorientation of material.

Supersonic trim of a transport aircraft

Consider the problem of increasing the stiffness of the low aspect ratio supersonic wing (previously shown in Figure 5) to reduce the trim drag that occurs during transition from subsonic to supersonic flight. A main contributor to this drag is the elastic deformation created by the deflection of elevons used to trim the airplane in pitch. The stiffness constraint to address this problem can be presented as a permissible value of the difference between wing torsional deformations in the subsonic and supersonic cruise modes. Figure 13 shows the distribution of this difference in torsional deformation for the supersonic TU - 144. The problem considered was the re-design of this wing (a problem for which the results were not implemented).

In general, the re-design of a wing with such stiffness requirements can be formulated as follows. Take as a measure of design stiffness the twist angle of the characteristic section previously defined. It is required to increase the design stiffness with a minimal weight increase. We are required to find the distribution of wing structural material that satisfies strength conditions for all loading

Twisting angles difference of wing aerofoil sections

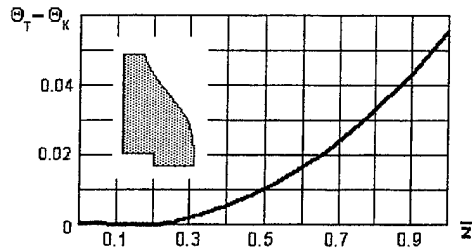


Figure 13 –Difference between supersonic and subsonic twisting angles conditions and provides a minimal difference between twisting of the characteristic section in the subsonic and supersonic flight phases and has minimum mass. To solve the design problem approximately, we do the following:

1. Assume that we know the optimal shape of the wing middle surface, including geometrical and aerodynamic twist and curvature of the middle surface so that we have minimum aerodynamic drag in supersonic cruise.
2. Determine loads for all remaining load cases. With these loads we find the distribution of material satisfying only strength requirements.
3. With subsonic and supersonic cruise (from Step 1) we find a design that has a minimal difference of torsional deformations of the characteristic section at subsonic and supersonic flight.

Additional thickness for upper skin [mm]

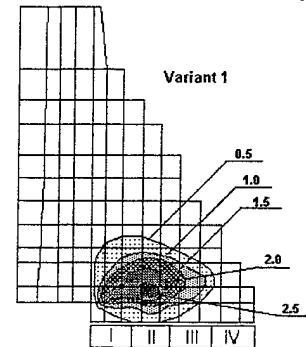


Figure 14 – Distribution of added mass on upper wing skins

The additional material can be added to skin panel and spar elements, but should first be placed in zones with large positive values of differences between Mohr integrals calculated for this problem.

There are other less effective ways to handle this problem. These include

1. Use a more powerful engine; however, in the cruise mode its additional thrust is not necessary; also it will not remove control problems that occur at transition to a subsonic speed.
2. Transfer part of the fuel to tail tanks elsewhere to displace the center of mass aft; this has several drawbacks. First of all, the useful internal volume of the airplane is decreased. For a fast fuel transfer, powerful (usually - some tens of kilowatts) fuel pumps are required with

high output fuel lines. This makes the airplane fuel system and power supply system much heavier and more costly. In addition, in an emergency situation we must decrease speed from supersonic to subsonic quickly. Then it is necessary to dump transferred fuel to prevent longitudinal instability of the airplane in the subsonic region because of the aft position of the center of mass. Despite these apparent disadvantages, this method is often used.

3. We can use canards. However, flight tests demonstrate that the efficiency at subsonic speed is poor and longitudinal balance problems remain. In addition, the spin property of the airplane with a canard configuration is worse than other airplanes. Also, an airplane with a canard configuration has poor dynamic stability since there is difficulty suppressing short-period oscillations.

The maximum additional material allowed is 1% of the original structural weight. Figures 14 and 15 show the results of placing additional material in the skin panels for stiffening such that the difference between the

This process is very rapid and has far reaching effects on the design.

Clearly, the simplicity of an effective optimization method can help a design team sort through feasible solutions to a serious design problem.

Conclusion

Our procedure, implemented by a PC software package known as DRACO developed by the first author or by alternative optimization techniques, provides the ability to perform high-level trades early in the design process. Two illustrative examples have been discussed to show the level of input detail required for the process. This approach also places structural topology with aeroelastic considerations on an equal, multi-disciplinary level with the usual aerodynamic and performance considerations that drive conceptual design. The results of this process can then be given to high fidelity optimization/analysis packages such as ASTROS for further refinement at the detailed design level - if the design actually progresses that far. The result will be a higher fidelity, higher quality effort capable of identifying structural and weight problems early, while providing reliable design information.

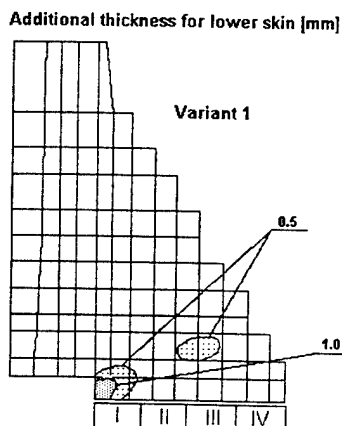


Figure 15 – Distribution of added mass in lower skins

subsonic and supersonic twist is minimized.

This study illustrates how optimality criteria, coupled with effective finite element analysis, can bring valuable information to the initial structural design effort. This information allows the design team the opportunity to review and debate design decisions made by configuration specialists early so that good decisions are made. Using this technique, it is no longer necessary to wait to consider requirements such as aeroelasticity or other stiffness related performance constraints.

References

1. L.A. Schmidt, Structural Synthesis-Its Genesis and Development, AIAA Journal, Vol. 19, No. 10, October 1981.
2. V. A. Komarov, "Design of Load-carrying Scheme of Airframe," Actual Problems of Aircraft Science and Engineering, Mashinostroyeniye, Moscow, 1984 (in Russian).
3. L.A. McCullers, "Automated Design of Advanced Composite Structures," Proceedings of the ASME Structural Optimization Symposium, AMD-Vol. 7, Nov. 1974, pp. 119-133.
4. Wilkinson, K. et al., "An Automated Procedure for Flutter and Strength Analysis and Optimization of Aerospace Vehicles, Vol. 1 - Theory and Application," AFFDL-TR-75-137, U.S.A.F. Flight Dynamics Laboratory, December 1975.
5. ASTROS
6. *New World Vistas-Air and Space Power for the 21st Century*, Aircraft and Propulsion, A Report to the Secretary of the Air Force, December 1995.
7. V. A. Komarov and T.A. Weisshaar, "Aircraft Structural Design - Improving Conceptual Design Fidelity," AIAA Paper 98-4885, 7th AIAA/USAF/NASA/ISSMO Symposium on Multidisciplinary Optimization, St. Louis, Mo., September 1998. Grose, Sensburg, O. and Tischler, V. "A Unique Design for a Diverging Flexible Vertical Tail," NATO/RTO Meeting, Ottawa, Canada, October 1999